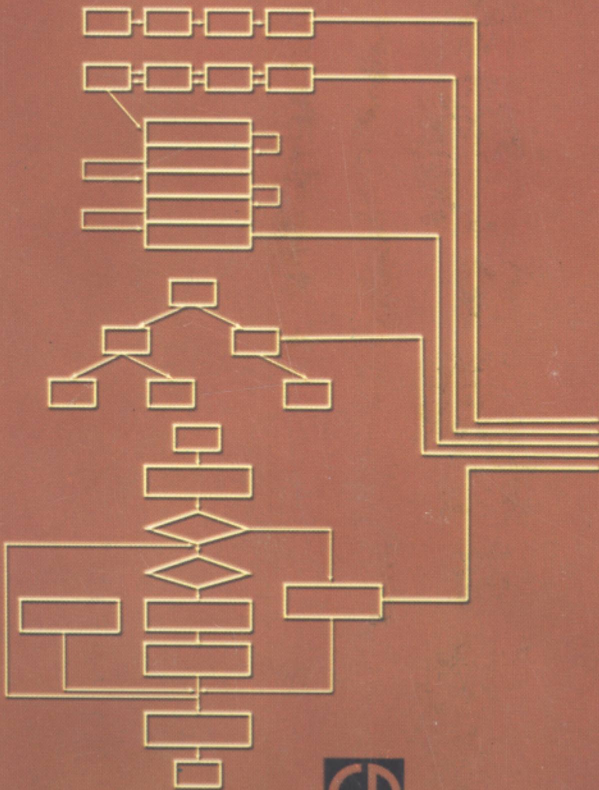
VỤ GIÁO DỤC CHUYÊN NGHIỆP

GIÁO TRÌNH NGÔN NGỮ LẬP TRÌNH C

SÁCH DÙNG CHO CÁC TRƯỜNG ĐÀO TẠO HỆ TRUNG HỌC CHUYÊN NGHIỆP







NHÀ XUẤT BẢN GIÁO DỤC

ThS. TIÊU KIM CƯƠNG

GIÁO TRÌNH

NGÔN NGỮ LẬP TRÌNH C

(Sách dùng cho các trường đào tạo hệ Trung học chuyên nghiệp)

NHÀ XUẤT BẢN GIÁO DỤC

$\frac{6T7}{GD - 04}$ 65 /115 - 04

Mã số: 6H159M4

Lời giới thiệu

Năm 2002, Vụ Giáo dục Chuyên nghiệp – Bộ Giáo dục và Đào tạo đã phối hợp với Nhà xuất bản Giáo dục xuất bản 21 giáo trình phục vụ cho đào tạo hệ THCN. Các giáo trình trên đã được nhiều trường sử dụng và hoan nghênh. Để tiếp tục bổ sung nguồn giáo trình đang còn thiếu, Vụ Giáo dục Chuyên nghiệp phối hợp cùng Nhà xuất bản Giáo dục tiếp tục biên soạn một số giáo trình, sách tham khảo phục vụ cho đào tạo ở các ngành : Điện – Điện tử, Tin học, Khai thác cơ khí. Những giáo trình này trước khi biên soạn, Vụ Giáo dục Chuyên nghiệp đã gửi đề cương về trên 20 trường và tổ chức hội thảo, lấy ý kiến đóng góp về nội dung đề cương các giáo trình nói trên. Trên cơ sở nghiên cứu ý kiến đóng góp của các trường, nhóm tác giả đã điều chỉnh nội dung các giáo trình cho phù hợp với yêu cầu thực tiễn hơn.

Với kinh nghiệm giảng dạy, kiến thức tích luỹ qua nhiều năm, các tác giả đã cố gắng để những nội dung được trình bày là những kiến thức cơ bản nhất nhưng vẫn cập nhật được với những tiến bộ của khoa học kỹ thuật, với thực tế sản xuất. Nội dung của giáo trình còn tạo sự liên thông từ Dạy nghề lên THCN.

Các giáo trình được biên soạn theo hướng mở, kiến thức rộng và cố gắng chỉ ra tính ứng dụng của nội dung được trình bày. Trên cơ sở đó tạo điều kiện để các trường sử dụng một cách phù hợp với điều kiện cơ sở vật chất phục vụ thực hành, thực tập và đặc điểm của các ngành, chuyên ngành đào tạo.

Để việc đổi mới phương pháp dạy và học theo chỉ đạo của Bộ Giáo dục và Đào tạo nhằm nâng cao chất lượng dạy và học, các trường cần trang bị đủ sách cho thư viện và tạo điều kiện để giáo viên và học sinh có đủ sách theo ngành đào tạo. Những giáo trình này cũng là tài liệu tham khảo tốt cho học sinh đã tốt nghiệp cần đào tạo lại, nhân viên kỹ thuật đang trực tiếp sản xuất.

Các giáo trình đã xuất bản không thể tránh khỏi những sai sót. Rất mong các thầy, cô giáo, bạn đọc góp ý để lần xuất bản sau được tốt hơn. Mọi góp ý xin gửi về : Công ty Cổ phần sách Đại học – Dạy nghề 25 Hàn Thuyên – Hà Nội.

VỤ GIÁO DỤC CHUYÊN NGHIỆP - NXB GIÁO DỤC

Mở đầu

Giáo dục chuyên nghiệp và dạy nghề là một trong những lĩnh vực cần được quan tâm hàng đầu trong tiến trình phát triển của một đất nước. Một trong những nhân tố đóng vai trò quyết định đến chất lượng của đào tạo đó là giáo trình dùng để giảng dạy trong nhà trường. Khác với giáo dục đại học, giáo dục chuyên nghiệp đòi hỏi phải có sự kết hợp nhuần nhuyễn giữa lí thuyết và thực hành. Những kiến thức lí thuyết đã học phải được áp dụng ngay trong thực tế thông qua các ví dụ cụ thể. Tuy nhiên, không vì thế mà nội dung lí thuyết bị cắt giảm đi, ngược lại cần phải đào sâu, mở rộng thêm để việc áp dụng của người học không máy móc, dập khuôn, mà phải sáng tạo và chủ động dựa trên những hiểu biết sâu sắc của bản thân, có như thế khi tốt nghiệp mới đáp ứng được những nhu cầu ngày càng cao của xã hội.

Ngôn ngữ lập trình C là một trong những ngôn ngữ khó sử dụng, đòi hỏi nhiều thời gian và công sức để có thể làm chủ nó, biến nó thực sự trở thành một công cụ đắc lực trong cuộc sống nghề nghiệp của mỗi người. Do đó, trong khuôn khổ 60 tiết, tác giả cố gắng biên soạn trên tinh thần ngắn gọn, dễ hiểu và đầy đủ dựa trên quan điểm dạy học tích cực – dạy học định hướng hành động. Các kiến thức trong toàn bộ giáo trình có mối liên hệ chặt chẽ, logic. Các nội dung đưa ra được minh họa cụ thể, trực quan và được phân tích sâu sắc nhằm giúp người học có thể hiểu kỹ hơn các tình huống có vấn đề và thường gặp trong công việc.

Toàn bộ giáo trình bao gồm bảy chương và bốn phụ lục chứa đựng tương đối đầy đủ các vấn đề cơ bản nhất của ngôn ngữ lập trình C, các loại ví dụ và bài tập chọn lọc cùng một số vấn đề liên quan, giúp người học có khả năng sử dụng thành thạo ngôn ngữ này trong việc giải quyết một số lớp bài toán thông dụng trong thực tế. Đầu mỗi chương, đều chỉ rõ các mục tiêu cụ thể cần đạt được của chương đó nhằm giúp người học có thể định hướng tốt hơn trong việc học cũng như giúp cho giáo viên có thể tham khảo trong quá trình giảng dạy của mình.

Giáo trình được biên soạn cho đối tượng chính là học sinh THCN, Kĩ thuật viên tin học, tuy nhiên nó cũng có thể là tài liệu tham khảo bổ ích cho bậc đại học và những người quan tâm.

Mặc dù đã cố gắng nhiều trong quá trình biên soạn giáo trình này, nhưng chắc chắn không tránh khỏi có những thiếu sót. Rất mong nhận được ý kiến đóng góp của độc giả và các đồng nghiệp để giáo trình ngày càng hoàn thiện hơn. Mọi ý kiến đóng góp xin gửi về Nhà xuất bản Giáo dục – 81 Trần Hưng Đạo, Hà Nội.

TÁC GIẢ

Chương 1

TỔNG QUAN VỀ NGÔN NGỮ LẬP TRÌNH C

MỤC TIÊU CỦA CHƯƠNG NÀY

- *Biết một vài nét lịch sử phát triển cũng như đặc điểm của ngôn ngữ lập trình C đồng thời hiểu và biết cách sử dụng các khái niệm cơ bản trong ngôn ngữ lập trình C.*
- *Biết cách sử dụng môi trường kết hợp của Turbo C để viết và chạy một chương trình C đơn giản theo cấu trúc chuẩn.*

1.1. LỊCH SỬ PHÁT TRIỂN VÀ ĐẶC ĐIỂM NGÔN NGỮ LẬP TRÌNH C

Tiền thân của ngôn ngữ lập trình C là ngôn ngữ BCPL (Basic Conbined Programming Language) do Martin Richards nghiên cứu. Ảnh hưởng của ngôn ngữ BCPL lên ngôn ngữ lập trình C gián tiếp thông qua ngôn ngữ B, do Ken Thompson viết năm 1970 cho hệ điều hành UNIX chạy trên họ máy tính PDP-7.

Nhu cầu cải tiến và phát triển cho UNIX đã thúc đẩy Dennis Ritchie và Brian Kernighan sáng tạo ra ngôn ngữ lập trình C ngay tại phòng thí nghiệm BELL (Hoa Kì) vào đầu những năm 70 nhằm mục đích ban đầu là phát triển một ngôn ngữ hệ thống mềm dẻo thay thế cho ngôn ngữ Assembly vốn nặng nề và "cứng nhắc" với các thiết bị phần cứng.

Ngôn ngữ lập trình C đặc biệt khác với ngôn ngữ BCPL và ngôn ngữ B ở chỗ: ngôn ngữ BCPL và ngôn ngữ B chỉ có duy nhất một kiểu dữ liệu là ***từ máy***, trong khi đó ngôn ngữ lập trình C đã có các đối tượng dữ liệu cơ bản như ***kí tự, các kiểu số nguyên và các kiểu số thực***. Đặc biệt ***con trỏ*** trong ngôn ngữ lập trình C tạo ra thêm được rất nhiều ưu việt.

Sau khi ra đời, đặc biệt thành công với hệ điều hành UNIX, ngôn ngữ lập trình C bắt đầu được phổ biến rộng rãi và người ta đã nhận thấy sức mạnh của nó. C là ngôn ngữ lập trình tương đối vạn năng, có mức độ thích nghi cao, mềm dẻo. Khác với ngôn ngữ Pascal, là ngôn ngữ lập trình có cấu trúc rất chặt chẽ và thường được dùng để giảng dạy về lập trình đặc biệt trong các trường đại học, thì ngôn ngữ lập trình C lại được sử dụng rộng rãi trong các lĩnh vực chuyên nghiệp vì tính hiệu quả và mềm dẻo của nó.

Vào những năm 80, do nhu cầu trong việc xử lí dữ liệu ngày một cao, các chương trình viết ra ngày một phức tạp, việc bảo dưỡng chương trình ngày một khó

khăn đã dẫn đến một phong cách lập trình mới – lập trình hướng đối tượng (*OOP – Object Oriented Programming*) xuất hiện và ngôn ngữ lập trình C bắt đầu được trang bị thêm khả năng lập trình hướng đối tượng, ngôn ngữ lập trình C^{++} ra đời từ đó và ngày càng chiếm ưu thế.

Hiện nay có rất nhiều bộ chương trình biên dịch (*Compiler*) và liên kết (*Link*) cho ngôn ngữ lập trình C của nhiều hãng khác nhau và mỗi bộ chương trình đều có những ưu, nhược điểm riêng. Xếp ở vị trí hàng đầu có thể kể đến *Turbo C* của hãng Borland, *MS C* của hãng Microsoft, *ZORTECH-C* của hãng SYSMANTEC. Phần mềm *Turbo C* được sử dụng khá rộng rãi vì nó cung cấp cho người dùng một thư viện khá đầy đủ các hàm vào ra, truy nhập và đồ hoạ. Tuy nhiên khả năng tối ưu mã của nó không bằng *MS C*. Trong giáo trình này chúng tôi sử dụng *Turbo C* do tính tiện lợi và phổ dụng của nó.

1.2. CÁC KHÁI NIỆM CƠ BẢN

1.2.1. Tập kí tự dùng trong ngôn ngữ lập trình C

Cũng như ngôn ngữ tự nhiên, mọi ngôn ngữ lập trình đều được xây dựng từ một bộ kí tự nào đó gọi là bảng chữ cái của ngôn ngữ. Các kí tự này được kết hợp với nhau theo nhiều cách, theo nhiều quy tắc khác nhau tạo nên các ***từ***, các ***từ*** lại được liên kết với nhau theo một quy tắc nào đó (*phụ thuộc vào các ngôn ngữ khác nhau*) để tạo thành các câu lệnh. Mỗi chương trình sẽ bao gồm nhiều câu lệnh được diễn đạt theo *logic* của một thuật toán nào đó để giải quyết bài toán đang xét. Ngôn ngữ lập trình C được xây dựng trên bộ kí tự sau đây:

- 26 chữ cái hoa A, B, C, ... Z và 26 chữ cái thường a, b, c, ... z của bộ chữ cái tiếng Anh.
- 10 chữ số 0, 1, 2, ... 9
- Các kí hiệu toán học: + - * / = ( )
- Kí tự gạch nối ' _ ' (chú ý khác với kí tự ' - ')
- Các kí hiệu đặc biệt khác đó là các kí tự : , . ; [ ] { } ? ! & \ % # $ ~ > < ^ " " ' ' và |.

Dấu cách ' ' (*kí tự trống*) được dùng như kí tự phân tách giữa các ***từ*** trong chương trình và không có ý nghĩa gì trong mỗi câu lệnh, các ***từ*** trong ngôn ngữ lập trình C có thể được phân tách bởi một hay nhiều dấu cách.

Khi viết chương trình ta không được phép sử dụng bất kì một kí tự nào khác ngoài các kí tự kể trên.

1.2.2. Từ khoá

Từ khoá là những từ ***dành riêng*** của ngôn ngữ lập trình C được định nghĩa trước với những ý nghĩa hoàn toàn xác định. Từ khoá thường được dùng để khai báo, định nghĩa các kiểu dữ liệu, định nghĩa ra các toán tử, các hàm và viết các câu lệnh... Do đó khi viết chương trình ta không thể dùng từ khoá để đặt tên cho các hằng, biến, mảng hay hàm ...

***Chú ý*: Trong ngôn ngữ lập trình C từ khoá bao giờ cũng được viết bằng chữ thường.**

Bảng 1.1: Các từ khoá thông dụng trong ngôn ngữ lập trình C

asm	do	goto	register	typedef
break	double	huge	return	union
case	else	if	short	unsigned
cdecl	enum	int	signed	void
char	extern	interrupt	sizeof	volatile
const	far	long	static	while
continue	float	near	struct	
default	for	pascal	switch	

Mỗi từ khoá sẽ có cú pháp sử dụng riêng và trong chương trình ta phải tuân thủ đúng theo cú pháp đó. Cách sử dụng của các từ khoá thông dụng nhất sẽ được lần lượt giới thiệu trong giáo trình.

1.2.3. Cách đặt tên trong ngôn ngữ lập trình C

Tên là một khái niệm rất quan trọng dùng để xác định các đại lượng khác nhau trong mỗi chương trình như tên hằng, tên biến, tên mảng, tên cấu trúc, tên con trỏ, tên tệp, tên nhãn, tên hàm... Mỗi tên trong ngôn ngữ lập trình C được quy ước là một dãy các kí tự chỉ bao gồm chữ cái, chữ số và dấu gạch nối. Kí tự đầu tiên của tên bắt buộc phải là một chữ cái hoặc dấu gạch nối.

Sau đây là một số ví dụ về tên:

Δ: Không hợp lệ vì không phải chữ cái, chữ số hoặc dấu gạch nối.
2_delta: Không hợp lệ vì bắt đầu bằng chữ số.
switch: Không hợp lệ vì trùng với từ khoá.
Del ta: Không hợp lệ vì có khoảng cách ở giữa.
Del-ta: Không hợp lệ vì sử dụng dấu gạch ngang '-'.
Del_ta: Hợp lệ.
_Modul_asm_1 : Hợp lệ.
Trung_gian_1: Hợp lệ ...

Khác với Pascal, tên trong ngôn ngữ lập trình C phân biệt giữa chữ hoa và chữ thường, ví dụ các tên *abc*, *Abc*, *ABc*, *ABC* ... là hoàn toàn khác nhau.

Một số lưu ý khi đặt tên:

- Không đặt tên trùng với từ khóa.
- Trong cùng một phạm vi và cùng thời gian tồn tại không được phép đặt hai tên trùng nhau. Có nghĩa là, trong cùng một ***khối lệnh*** (*xem mục 1.2.5*) không được phép khai báo hai tên trùng nhau.
- Tên phải phản ánh được bản chất của đối tượng được đặt tên. Ví dụ khi khai báo một biến dùng để chứa kích thước của một mảng ta nên dùng *ArraySize* để nhấn mạnh thay vì dùng *arraysize* hoặc một tên hoàn toàn không gợi nhớ như *as* ...
- Có nhiều quy ước đặt tên khác nhau, ví dụ như các lập trình viên của MicroSoft Windows và OS/2 thì thường đặt tên theo „*cách viết kiểu Hungary*". Theo cách viết này mỗi tên phải được ghi thêm đằng trước một tiền tố phân biệt kiểu và phạm vi của biến đó. Ví dụ, một biến toàn cục kiểu số nguyên dùng để chứa

kích thước của một mảng có thể viết như sau: *giArraySize* (*g viết tắt của global, i viết tắt của integer*). Theo cách viết này chương trình trở nên tương đối sáng sủa, tuy nhiên với các kiểu dữ liệu tự xây dựng thì cách viết này tỏ ra tương đối khó sử dụng. Trong giáo trình này, chúng tôi quy ước đặt tên như sau: mọi tên chung, toàn cục (*như tên hàm, tên cấu trúc...*) được viết bằng cả hai kiểu chữ, bắt đầu bằng một chữ *g* (*viết tắt của global*), các từ sẽ phân tách nhau bởi kí tự hoa. Mọi tên địa phương của hàm được viết bằng cả hai kiểu chữ, các từ cũng phân tách nhau bởi kí tự hoa. Các hằng số được viết bằng chữ in. Các hằng kiểu ***enum*** được viết bằng cả hai kiểu chữ.

- Độ dài tối đa mặc định của một tên trong ngôn ngữ lập trình C là *32*, tuy nhiên ta có thể thay đổi lại bằng một giá trị từ 1 đến 32 trong *Option->Compiler -> Source->Identifier*. Nếu giá trị này vượt quá 32 thì chỉ 32 kí tự đầu tiên được được chấp nhận.

1.2.4. Kiểu dữ liệu

1. Các kiểu dữ liệu có sẵn

Turbo C định nghĩa sẵn 4 kiểu dữ liệu cơ bản đó là: ***char***, ***int***, ***float*** và ***double***. Chúng được mô tả chi tiết trong bảng sau:

Bảng 1.2. Các kiểu dữ liệu trong ngôn ngữ lập trình C

Kiểu	Mô tả	Phạm vi biểu diễn	Kích thước
char	Kiểu kí tự	-128 đến 127	1 byte
int	Kiểu số nguyên	-32768 đến 32767	2 bytes
float	Kiểu số thực dấu phẩy động độ chính xác đơn	±3.4E-38 đến ± 3.4E+38	4 bytes
double	Kiểu số thực dấu phẩy động độ chính xác kép	±1.7E-308 đến ± 1.7E+308	8 bytes

Một số điểm cần lưu ý:

- Số thực kiểu *float* có độ chính xác là *6* chữ số sau dấu chấm thập phân, còn số thực kiểu *double* có độ chính xác là *15* chữ số sau dấu chấm thập phân, do vậy khi sử dụng nếu yêu cầu giá trị lớn, độ chính xác cao thì nên dùng *double*, ngược lại chỉ nên dùng *float*.

- Mỗi kiểu dữ liệu cơ bản trên lại có thể kết hợp với một hoặc nhiều "tiền tố" sau đây: ***short***, ***long***, ***signed*** (*ngầm định đối với char và int*) và ***unsigned***, để thay đổi phạm vi biểu diễn của mỗi kiểu dữ liệu đó. Một số kiểu kết hợp thông dụng có thể được mô tả thông qua bảng sau:

Bảng 1.3. Một số kiểu dữ liệu thông dụng có sử dụng thêm *tiền tố* trong C

Kiểu dữ liệu	Phạm vi biểu diễn	Kích thước
unsigned char	0 đến 255	1 byte
unsigned int hay unsigned	0 đến 65 535	2 bytes
short int	-32 768 đến 32 767	2 bytes
long int hay long	-2147483648 đến 2147483647	4 bytes
unsigned long int hay unsigned long	0 đến 4294967295	4 bytes
long double	±3.4E-4932 đến ± 1.1E+4932	10 bytes

2. Kiểu enum

Kiểu *enum* trong ngôn ngữ lập trình C là một loại kiểu liệt kê dùng để khai báo các biến chứa các đối tượng kiểu đếm được có giá trị thuộc một miền thứ tự được chỉ rõ trong lúc khai báo. Để tạo ra một dữ liệu kiểu *enum* ta sử dụng câu lệnh có cú pháp sau đây:

enum ten_kieu {danh sách các phần tử};

Trong đó ***ten_kieu*** là tên của kiểu dữ liệu liệt kê mới vừa được tạo ra, ***danh sách các phần tử*** là các giá trị liệt kê mà các biến có thể nhận được, các phần tử phân cách nhau bởi dấu phẩy.

Ví dụ 1-1. Để làm việc với các ngày trong tuần ta có thể dùng kiểu ***WeekDay*** như sau:

enum WeekDay {SUNDAY, MONDAY, TUESDAY, WEDSDAY, THURSDAY, FRIDAY, SATURDAY} *Day1*;

enum WeekDay *Day2*;

Khi đó *Day1* và *Day2* là các biến kiểu ***WeekDay*** và chúng có thể nhận các giá trị đã được liệt kê. Các câu lệnh sau đây là hợp lệ:

Day1 = SUNDAY;
Day2 = FRIDAY;

Thực chất các biến kiểu enum trong ngôn ngữ lập trình C được coi là các biến nguyên, chúng được cấp phát *2 bytes* bộ nhớ và có thể nhận một giá trị nguyên nào đó. Mỗi khi một dữ liệu kiểu enum được định nghĩa ra thì các phần tử trong ***danh sách các phần tử*** sẽ được gán cho các giá trị nguyên liên tiếp bắt đầu từ 0. Ví dụ như kiểu ***WeekDay*** ở trên thì SUNDAY sẽ có giá trị 0, MONDAY sẽ có giá trị 1... Do đó hai câu lệnh sau đây có kết quả giống nhau:

Day1 = 0; và *Day1* = SUNDAY;

3. Các kiểu tự định nghĩa

Trong ngôn ngữ lập trình C ta có thể tự định nghĩa ra các kiểu dữ liệu của riêng mình bằng cách thêm từ khoá *typedef* vào trước một khai báo nào đó.

Ví dụ 1-2. Định nghĩa kiểu bằng *typedef*.

Để khai báo một biến nguyên có tên là *nguyen* ta có thể viết như sau:

int nguyen;

Nhưng nếu ta thêm từ khoá *typedef* vào trước của khai báo đó:

typedef int nguyen;

Thì lúc này *nguyen* đã trở thành một kiểu dữ liệu mới và câu lệnh sau đây là hoàn toàn đúng: ***nguyen i, j*** ; tương tự ta cũng có thể định nghĩa ra một kiểu dữ liệu mới có tên là ***MangNguyen50*** dùng để khai báo các biến mảng nguyên có kích thước là 50 như sau:

typedef int MangNguyen50[50];

Sau câu lệnh này ***MangNguyen50*** sẽ trở thành một kiểu dữ liệu mới và ta có thể dùng nó để khai báo cho các biến tương tự như việc khai báo cho các biến có kiểu định sẵn. Câu lệnh sau sẽ tạo ra hai biến kiểu ***MangNguyen50*** là *m1* và *m2*, mỗi biến sẽ là một mảng kiểu số nguyên có kích thước là 50:

MangNguyen50 m1, m2;

1.2.5. Khối lệnh

Khối lệnh là một dãy các câu lệnh nằm trong khối bao bởi dấu "{"và dấu "}". Sau đây là một ví dụ.

Ví dụ 1-3. Cấu trúc của một khối lệnh.

```
{
   /* Các câu lệnh nằm ở đây*/ (1)
   int a;
   a=117;
   ...
}
```

Chú ý:

1. Trong chương trình C, mỗi khối lệnh về mặt logic được xem như một câu lệnh riêng lẻ, có nghĩa là trong chương trình, cứ chỗ nào đặt được một câu lệnh thì ta cũng có quyền đặt khối lệnh vào đó. Hiểu không rõ về khối lệnh là một lỗi thường rất hay mắc phải của những người mới bắt đầu học lập trình. Trong ngôn ngữ lập trình C cũng như các ngôn ngữ lập trình bậc cao khác, có những cấu trúc mà sau nó chỉ được phép đặt một câu lệnh mà thôi, ví dụ như cấu trúc ***if ... else***, cấu trúc ***for***, cấu trúc ***while*** ... nhưng thông thường sau các cấu trúc này ta cần phải thực hiện liên tiếp nhiều câu lệnh khác nhau mới đáp ứng được yêu cầu. Để giải quyết vấn đề này ta chỉ cần đưa tất cả các câu lệnh đó vào trong một khối lệnh và lúc này về mặt logic chúng sẽ được xem như là một lệnh duy nhất. Nếu ta không đưa vào trong khối lệnh sẽ dẫn đến hoặc chương trình chạy sai, hoặc sẽ báo lỗi.

2. Bên trong một khối lệnh lại có thể chứa các khối lệnh khác, sự lồng nhau này là không hạn chế.

3. Mỗi thân hàm thực chất là một khối lệnh dùng để bao các khối lệnh khác trong nó (*các vấn đề về hàm sẽ được trình bày trong chương 4*).

1.2.6. Biến và các đặc trưng của biến

1. Khái niệm:

Biến là vùng nhớ được cấp phát dùng để lưu trữ giá trị cho một kiểu dữ liệu nào đó tại một thời điểm nhất định và nó được truy xuất thông qua một tên đã được khai báo cho biến đó. Một biến trong ngôn ngữ lập trình C trước khi được sử dụng thì nó phải được khai báo ở ***đầu mỗi khối lệnh*** (*trước bất cứ câu lệnh nào khác*) theo cú pháp chung như sau:

Kiểu_dữ_liệu Tên_biến ;

---

[1] Trong ngôn ngữ lập trình C, tất cả các câu đặt trong cặp dấu /* */ được coi là những chú thích và được trình biên dịch bỏ qua.

Trong đó ***Kiểu_dữ_liệu*** sẽ xác định kiểu của dữ liệu mà biến lưu trữ. *Tên_biến* là một định danh được gán cho vùng nhớ chứa biến và dùng để truy xuất giá trị của biến. Dấu ';' để đánh dấu sự kết thúc của câu lệnh. Ví dụ khi ta có khai báo sau đây:

int SoVongLap;

thì khi biên dịch máy sẽ cấp cho ta một khoảng nhớ có độ dài 2 bytes có tên là *SoVongLap* dùng để lưu trữ các giá trị cho một số nguyên nào đó.

Trong một dòng lệnh ta hoàn toàn có thể khai báo nhiều biến cùng kiểu bằng cách phân tách các biến đó bởi các dấu ***phẩy*** theo cú pháp sau:

Kiểu_dữ_liệu Biến1, Biến2, ..., Biến_n ;

Ví dụ 1-4: Câu lệnh float a, b, c=10, d; sẽ khai báo ra *4* biến kiểu số thực a, b, c và d, trong đó chỉ có duy nhất biến c được *khởi gán giá trị ban đầu* là 10.

2. Đặc trưng của biến

Vị trí khai báo của biến: Đây là đặc trưng rất quan trọng của một biến, nó sẽ xác định ***phạm vi sử dụng*** và ***thời gian tồn tại*** của biến đó trong chương trình. Trong ngôn ngữ lập trình C phân biệt hai loại vị trí khai báo cho biến, đó là:

a) Nếu biến được khai báo ở bên ngoài các khối lệnh (*nghĩa là ở bên ngoài các hàm*) thì nó sẽ được gọi là ***biến ngoài*** hay còn gọi là ***biến toàn cục***. Phạm vi sử dụng (*phạm vi hoạt động*) của nó sẽ có giá trị từ vị trí khai báo cho đến hết tệp chương trình. Nghĩa là nó có thể được truy xuất từ bất cứ hàm nào bắt đầu từ vị trí khai báo cho đến hết tệp chương trình. Còn thời gian tồn tại (*thời gian được cấp phát bộ nhớ*) của biến là suốt thời gian mà chương trình làm việc. Nghĩa là giá trị của biến (*nếu có*) sẽ được lưu giữ trong suốt thời gian mà chương trình hoạt động. Một biến ngoài có thể được ***khởi gán một lần lúc dịch chương trình***, nếu không được khởi gán máy sẽ mặc định gán cho giá trị 0 hoặc NULL (*nếu là con trỏ*).

b) Nếu biến được khai báo bên trong một khối lệnh (*có nghĩa là bên trong các hàm*) thì nó được gọi là ***biến trong*** hay còn gọi là ***biến cục bộ*** hoặc ***biến tự động***. Phạm vi sử dụng của biến là bên trong khối lệnh mà nó được khai báo, nghĩa là nó chỉ có thể được truy xuất bên trong khối lệnh đó mà thôi. Còn thời gian tồn tại của biến (*thời gian được cấp phát bộ nhớ*) là bắt đầu từ khi máy làm việc với khối lệnh cho đến khi ra khỏi khối lệnh đó. Có nghĩa là khi ra khỏi khối lệnh, vùng nhớ được cấp phát cho biến sẽ bị xoá, và do đó các giá trị của biến cũng sẽ mất đi. Một biến trong (*không áp dụng cho mảng*) nếu không được khởi gán một giá trị nào đó thì biến đó hoàn toàn không xác định (*nhận một giá trị ngẫu nhiên sẵn có trong bộ nhớ lúc được cấp phát*).

Ví dụ 1-5. Để hiểu rõ hơn, hãy xét đoạn chương trình sau đây:

```
...
{       /*Khối lệnh 1*/
        int a=10, b;
        {       /*Khối lệnh 2*/
                int a, c;
                a=100; /* Gọi biến a của khối lệnh 2*/
                b=1000; /* Gọi biến b của khối lệnh 1*/
```

```
        }
        a=200; /* Gọi biến a của khối lệnh 1*/
        c=1999; /* Sai do phạm vi của biến c chỉ trong phạm vi khối lệnh 2*/
        ...
    }
    ...
```

Đoạn chương trình gồm có hai khối lệnh lồng nhau, khối lệnh ngoài khai báo hai biến nguyên *a* và *b*, khối lệnh trong khai báo hai biến nguyên *a* và *c*. Khi đó làm thế nào để phân biệt được biến *a* ở khối lệnh trong và biến *a* ở khối lệnh ngoài? Thực chất trong trường hợp này máy sẽ cấp phát hai khoảng nhớ khác nhau cho hai biến này, phạm vi hoạt động và thời gian tồn tại của chúng cũng khác nhau. Biến *a* ở khối lệnh trong có phạm vi hoạt động tại các câu lệnh của khối lệnh trong và nó chỉ tồn tại trong thời gian máy làm việc với khối lệnh này, ra khỏi khối lệnh trong biến *a* ở khối lệnh trong sẽ bị xoá. Còn phạm vi hoạt động của biến *a* ở khối lệnh ngoài bao gồm các câu lệnh bên trong khối lệnh ngoài nhưng không thuộc khối lệnh trong, việc thay đổi giá trị của biến *a* ở khối lệnh ngoài không ảnh hưởng gì đến giá trị của biến *a* ở khối lệnh trong và ngược lại. Phạm vi hoạt động của biến *b* thì gồm cả các câu lệnh của khối lệnh ngoài và khối lệnh trong (*do không có biến b ở khối lệnh trong*) và thời gian tồn tại của nó là trong suốt thời gian máy làm việc với hai khối lệnh này. Còn phạm vi hoạt động của biến *c* chỉ bao gồm trong các câu lệnh thuộc khối lệnh trong mà thôi, ra khỏi khối lệnh trong biến *c* không còn tồn tại nữa.

Loại biến: Mỗi biến sau khi khai báo còn được đặc trưng bởi các từ khoá đi kèm phía trước như ***static***, ***auto***, ***extern***, ***register***, ***const*** và ***volatile***.

Từ khoá ***auto*** dùng để chỉ rõ tính cục bộ của một biến được khai báo bên trong hàm. Vì các biến này đương nhiên là cục bộ cho nên từ khoá này trong ngôn ngữ lập trình C ít được dùng. Từ khoá ***extern*** được sử dụng khi một chương trình được viết trên nhiều tệp, cách sử dụng từ khoá này sẽ được trình bày chi tiết trong phụ lục I. Từ khoá ***register*** dùng để xác định một biến cục bộ có thể được lưu trữ trong các thanh ghi ***SI*** hoặc ***DI*** [(2)], khi các thanh ghi này bận thì các biến này được lưu trữ như các biến cục bộ khác. Do đó biến ***register*** có thể được dùng làm biến điều khiển để tăng tốc độ thực hiện của các vòng lặp. Từ khoá ***volatile*** dùng để báo cho Turbo C biết giá trị của biến có thể bị thay đổi theo một cách nào đó không được mô tả rõ trong chương trình và thường được sử dụng trong lập trình C nâng cao, do đó không được đề cập đến trong giáo trình này. Một từ khoá quan trọng mà ta cần nắm vững đó là từ khoá ***static***. Khi từ khoá ***static*** được đặt trước một khai báo của biến ngoài thì ta có ***biến tĩnh ngoài***, khi từ khóa này đặt trước một khai báo của biến trong (*biến cục bộ*) thì ta có ***biến tĩnh trong***. Cả biến tĩnh trong và biến tĩnh ngoài đều có thời gian tồn tại là trong suốt thời gian mà chương trình hoạt động, có nghĩa là nó được cấp phát bộ nhớ từ khi chương trình chạy cho đến khi kết thúc chương trình và do đó giá trị được lưu giữ trong các biến đó không mất đi trong suốt thời gian chương trình hoạt động. Tuy nhiên, phạm vi hoạt động của biến tĩnh trong chỉ giới hạn trong phạm vi khối lệnh mà nó được khai báo, còn phạm vi hoạt động của biến tĩnh ngoài được tính từ khi khai báo cho đến hết tệp chương trình. Nếu chương trình chỉ viết trên một tệp thì phạm vi hoạt động của ***biến tĩnh ngoài***

---

[2] Thanh ghi có thể hiểu là bộ nhớ đặc biệt bên trong bộ vi xử lí.

và ***biến ngoài*** là như nhau. Cả biến tĩnh trong và biến tĩnh ngoài đều có thể được ***khởi đầu một lần lúc dịch chương trình***. Nếu không sẽ nhận giá trị *0* hoặc *NULL*.

Chú ý

- Biến tĩnh ngoài không thể mở rộng sang tệp khác bằng từ khoá ***extern***.

- Nếu một biến ngoài được khai báo ở đầu chương trình (*trước tất cả các hàm*) thì nó có thể được sử dụng bởi bất kì hàm nào trong chương trình với điều kiện trong hàm đó không có biến cục bộ trùng tên với nó được khai báo.

- Nếu chương trình được viết trên nhiều tệp và các tệp được dịch độc lập thì phạm vi sử dụng của biến ngoài có thể mở rộng từ tệp này sang tệp khác bằng từ khoá ***extern*** (*xem phụ lục I*).

1.2.7. Hằng

1. Các loại hằng

Hằng là các đại lượng mà giá trị của nó không thay đổi trong quá trình thực hiện chương trình. Trong ngôn ngữ lập trình C có các loại hằng sau:

Hằng nguyên

Là đại lượng có giá trị từ *-32768* đến *32767*. Có thể biểu diễn một hằng nguyên dưới dạng thập phân, bát phân (*cơ số 8*) hay thập lục phân (*cơ số 16*). Ngôn ngữ lập trình C quy định biểu diễn một số dưới dạng bát phân bằng cách thêm 0 (*số không*) ở đầu, dưới dạng thập lục phân bằng cách thêm *0x*.

Ví dụ 1-6. Các loại hằng nguyên.

229 là một hằng nguyên hệ thập phân.

0345 là một hằng nguyên hệ bát phân, giá trị của nó trong hệ 10 là $3*8^2+4*8^1+5*8^0=229$.

0xA9 là một hằng nguyên hệ thập lục phân, giá trị của nó trong hệ 10 là $10*16^1+9*16^0=169$.

Hằng long

Giống như hằng nguyên, khác ở chỗ có thêm chữ *L* hoặc *l* ở đằng sau để biểu thị đó là hằng có giá trị *long*. Ví dụ: *198l*, *1234562L*.

Một hằng số nguyên vượt ra ngoài phạm vi cho phép được ngầm hiểu là hằng *long*.

Hằng số thực

Có hai cách viết giá trị của một hằng số thực. Trong cách viết thông thường, hằng được viết giống như trong thực tế nhưng thay dấu phẩy thập phân bằng dấu chấm. Ví dụ *359.12*, *-415.16*. Cách viết thứ hai là theo ký pháp khoa học. Theo cách viết này hằng được viết gồm hai phần: phần định trị có dạng *mm.nnn* và phần mũ phân tách với phần định trị bởi kí tự *e* hoặc *E*.

Ví dụ: *12.3E+3* giá trị của hằng này bằng $12.3 * 10^3=12300.0$; phần định trị (*12.3*) là số thực; phần mũ (*+3*) là số nguyên. Vì chúng ta có thể tăng giảm giá trị của phần mũ và tương ứng dịch chuyển dấu chấm thập phân trong phần định trị nên số thực được viết dưới dạng này còn có tên gọi là số thực dấu phẩy động, còn cách viết trước đó tương ứng với tên gọi số thực dấu phẩy tĩnh.

Chú ý: Khi viết hằng dấu phẩy tĩnh (*dạng thập phân*) thì phần nguyên hay phần thập phân có thể vắng mặt nhưng dấu chấm không được phép vắng mặt:

Ví dụ: *.25* hay *39.* là các hằng dấu phẩy tĩnh.

Hằng kí tự

Là một kí tự riêng biệt được đặt giữa hai dấu nháy đơn, ví dụ *'A'*, *'b'*. Giá trị của hằng kí tự chính là mã *ASCII* của kí tự đó [3].

Ví dụ: Hằng kí tự *'A'* có giá trị 65, hằng kí tự *'d'* có giá trị 100.

Hằng kí tự về thực chất có thể coi nó như một giá trị nguyên. Do đó nó cũng có thể tham gia vào các phép toán số học như mọi số nguyên khác. Ví dụ ta có thể viết:

'a'-'A' biểu thức này thực chất là *97- 65 = 32 = ' '* (*kí tự trống*).

Hằng kí tự còn có thể được viết là $'\backslash x_1x_2x_3'$, trong đó x_1, x_2, x_3 là các chữ số của hệ đếm cơ số 8 mà giá trị của $x_1x_2x_3$ bằng mã *ASCII* của kí tự đó.

Ví dụ *'\142'* là hằng kí tự *'b'*, *'\101'* là hằng kí tự *'A'*.

Có một số hằng kí tự đặc biệt được viết theo quy ước trong bảng sau:

Bảng 1.4. Quy ước viết một số kí tự đặc biệt trong ngôn ngữ lập trình C

Viết	Kí tự	Diễn giải
'\''	'	Dấu nháy đơn
'\,'	,	Dấu phẩy
'\"'	"	Dấu nháy kép
'\\'	\	Dấu gạch chéo ngược
'\n'	\n	Kí tự xuống dòng
'\0'	\0	Kí tự NULL
'\t'	Tab	Kí tự Tab
'\b'	Backspace	Kí tự Backspace
'\r'	CR	Kí tự trở về đầu dòng

Hằng xâu kí tự

Là một dãy kí tự được đặt trong cặp dấu nháy kép " ".

Ví dụ:

"Thành phố Hồ Chí Minh", "Wellcome to Hanoi", "" (*xâu rỗng*)… Xâu kí tự được lưu trữ trong máy dưới dạng một mảng kiểu ***char***. Trình biên dịch sẽ tự động thêm kí tự NULL (*kí tự '\0'*) vào cuối mỗi xâu để đánh dấu sự kết thúc của một xâu.

Chú ý:

Cần phân biệt giữa kí tự *'d'* và xâu *"d"*. *'d'* là hằng kí tự được lưu trữ trong một byte còn *"d"* là một xâu kí tự được lưu trữ trong một mảng gồm hai phần tử là *'d'* và *'\0'*.

2. Tên hằng và biến hằng

Biến hằng là một loại biến mà giá trị của nó không thể thay đổi trong lúc chạy chương trình [4]. Còn tên hằng là một loại hằng được định nghĩa bằng chỉ thị *#define*.

---

[3] Bảng mã *ASCII* là bảng mã chuẩn dùng để mã hoá cho các kí tự. Xem phụ lục IV.

a) Tên hằng

Được định nghĩa theo một trong hai cú pháp sau:

```
#define <Ten>  <Gia_tri>
#define Ten(Danh_sach_doi) Bieu_thuc
```

Cấu trúc thứ nhất định nghĩa một ***tên hằng*** có tên là *Ten* có giá trị là *Gia_tri*. *Gia_tri* ở đây có thể là một dãy kí tự, một giá trị số, một tên hàm… Khi biên dịch, chương trình dịch sẽ thay thế các lần xuất hiện của *Ten* bằng *Gia_tri* tương ứng đã được định nghĩa.

Ví dụ 1-7. Xét đoạn chương trình sau:

```
#define begin {
#define end }
#define MAX 20000
#define chxh "Cong hoa xa hoi chu  nghia Viet Nam"
#define in printf
int main()
begin
   in("\nChuoi dinh nghia la: %s", chxh);
   in("\n Gia tri Max= %d", MAX);
   return 0;
end
```

Khi biên dịch hàm *main()* được thay thế như sau:

```
int main()
{
   printf("\nChuoi dinh nghia la: %s","Cong hoa xa hoi chu nghia Viet\ (5)
   Nam");
   printf("\n Gia tri Max= %d", 20000);
   return 0;
}
```

Chương trình biên dịch sẽ thay thế các tên hằng *begin, end, MAX, chxh, in* bằng giá trị được định nghĩa tương ứng là *'{', '}', 20000, "Cong hoa xa hoi chu nghia Viet Nam"* và *printf*.

Cấu trúc thứ hai dùng để định nghĩa các *Macro* cho chương trình. Ví dụ ta có thể định nghĩa ra một Macro dùng để tính diện tích của một hình chữ nhật có hai cạnh tương ứng là A và B như sau:

Ví dụ 1-8. Viết *Macro* tính diện tích hình chữ nhật.

```
#include "stdio.h"
#define DienTich(A,B)  (A)*(B)
int main()
```

---

[4] Biến hằng hay được sử dụng khi trong chương trình ta cần đến các biến mà giá trị của nó không thể bị thay đổi trong suốt thời gian chương trình hoạt động.

[5] Trong ngôn ngữ lập trình C, để viết một chuỗi kí tự trong dấu " " trên nhiều dòng thì phải đặt dấu '\' vào cuối dòng trước đó.

```
{
    int a=10, b=20;
    float c=30, d=4;
    printf("Dien tich cua hinh chu nhat co canh a, b la:  %d", DienTich(a, b));
    printf("Dien tich cua hinh chu nhat co canh c, d la: %.2f", DienTich(c, d));
    return 0;
}
```

Khi biên dịch, trình biên dịch sẽ thay thế các câu lệnh:

DienTich(a, b); thành *a*b* và *DienTich(c, d)*; thành *c*d* và ta có kết quả đúng của diện tích các hình chữ nhật đó.

Chú ý:

- Một định nghĩa dài có thể được tiếp tục ở dòng sau bằng cách đặt dấu '\' vào cuối của dòng trước.

- Phạm vi của biến hằng được định nghĩa bởi *#define* là từ khi nó được định nghĩa cho đến cuối tệp gốc. Tuy nhiên một biến hằng *Ten* cũng có thể được định nghĩa lại sau câu lệnh *#undef Ten*.

- Phép thay thế không thực hiện cho các hằng chuỗi kí tự. Ví dụ như tên hằng *chxh* đã được định nghĩa ở trên nhưng nếu nó được đặt trong hằng xâu kí tự sau "*Toi noi **chxh** voi moi nguoi*" thì *chxh* trong xâu này không thể được thay thế bằng giá trị đã được định nghĩa.

- Khi định nghĩa các Macro bằng *#define* cần lưu ý là phải luôn đặt các đối số của *Bieu_thuc* trong cặp dấu ngoặc ( ). Ví dụ ta xét lại Macro dùng để tính diện tích của một hình chữ nhật đã có ở trên, nếu ta viết lại như sau:

```
#define DienTich(A, B) A*B
```

thì khi gặp câu lệnh sau:

DienTich(10+8,7); trình biên dịch sẽ thay thế bằng biểu thức *10+8*7* do đó kết quả sẽ bị sai. Nhưng nếu ta viết lại là :

#define DienTich(A, B) (A)*(B) khi thực hiện ta sẽ nhận được kết quả là: (10+8)*(7) và lúc này chương trình cho kết quả đúng.

b) Biến hằng

Được định nghĩa bằng từ khoá const với cú pháp như sau:

const Kieu TenBienHang = giá_trị;

Ví dụ 1-9. Cách khai báo cho biến hằng.

```
const  int  MAXLINE  = 100;
const  char NEWLINE  =  '\n';
const  char DHBK[18] = "DAI HOC BACH KHOA";
```

Về mặt ý nghĩa, các câu lệnh bắt đầu bằng *const* xác định một biến có giá trị không thay đổi (*biến hằng*). Nghĩa là mọi cố gắng nhằm thay đổi giá trị của các biến này sau khi khai báo đều không hợp lệ và gây ra lỗi biên dịch.

1.2.8. Câu lệnh

Câu lệnh là đơn vị nhỏ nhất của một chương trình máy tính, có nghĩa là tất cả các chương trình đều phải được xây dựng lên từ tập hợp các câu lệnh theo một thứ tự logic nào đấy. Mỗi ngôn ngữ lập trình sẽ quy ước viết các câu lệnh khác nhau. Trong ngôn ngữ lập trình C mỗi câu lệnh phải được kết thúc bằng một dấu " ; " (*ngoại trừ các chỉ thị tiền xử lí như #define, #include...*) và chúng có thể được viết trên một hoặc nhiều dòng. Tuy nhiên khi viết các câu lệnh trên nhiều dòng cần phải tuân theo một số quy tắc nhất định.

Quy tắc viết một câu lệnh: Giữa các ***từ*** trong một câu lệnh có thể đặt một hoặc một số các dấu cách ' ' hoặc dấu xuống dòng (*kí tự '\n'*). Điều này có nghĩa là ta không được phép bẻ gãy một ***từ*** trên nhiều dòng hoặc làm gián đoạn một ***từ*** bởi dấu cách.

Từ là một dãy kí tự viết liền nhau mang một ý nghĩa nhất định. Trong ngôn ngữ lập trình C có các loại ***từ*** sau đây:

a) ***Các hằng.*** Ví dụ *'K', -100.00, "Hello !"*

b) ***Các tên.*** Ví dụ *HoanVi, DienTich...*

c) ***Các từ khoá.*** Ví dụ *for, if, else ...*

d) ***Các dấu phép tính.*** Ví dụ *(, &, ++, --, == ...*

e) ***Một số dấu chức năng.*** Ví dụ *;, }, :...*

Sau đây là một số ví dụ về cách viết của các câu lệnh.

Ví dụ 1-10. Cách viết một câu lệnh.

```
S + = a; /* Sai vì += là một từ, do đó không được phép đặt dấu cách vào
giữa*/
for ( i=0 ; i<          /* Đúng vì các từ vẫn đảm bảo tính liên tục của các từ*/
      2 ; ++
           i);
dou                     /*Sai vì từ khoá double bị bẻ gẫy*/
ble x=200.0, y=299.23;  /* thành hai dòng, hằng xâu trong */
printf("\n x= %10.2f
y= %10.2f", x,y);       /*Sai vì hằng xâu bị bẻ thành hai dòng*/
```

Chú ý:

- Đối với câu lệnh *#include* [6] và *#define*, quy tắc nói trên không hoàn toàn đúng, giữa dấu #, từ khoá *include* và tên tệp có thể đặt một số bất kì khoảng cách nhưng phải trên một dòng. Tương tự với *#define*.

- Với một hằng xâu, ta có thể viết trên nhiều dòng khác nhau nhưng phải thêm một dấu '\' vào cuối dòng trước.

1.2.9. Vào / ra

Vào / ra là các thao tác cơ bản để giao tiếp giữa máy tính với thế giới bên ngoài (*các thiết bị ngoại vi*). Các thiết bị ngoại vi thì rất đa dạng và phát triển không ngừng, cho nên một trong những hướng đi cho vấn đề này là tạo ra các giao

---

[6] Chỉ thị này sẽ được giới thiệu trong mục 1.3.

điện không phụ thuộc phần cứng (*như cơ chế máy ảo trong Java*). Trong ngôn ngữ lập trình C, để tạo ra cơ chế *vào / ra* mềm dẻo người ta không trao đổi dữ liệu trực tiếp giữa chương trình với các thiết bị ngoại vi mà thông qua các kênh trung gian, các kênh này được xem như là các *thiết bị logic* và được gọi là các *dòng xuất nhập*. Các *dòng xuất nhập* thực chất là các vùng đệm được gán tương ứng với các thiết bị vật lí thực sự và chúng được tự động mở mỗi khi môi trường kết hợp Turbo C hoạt động (*xem thêm chương 5, mục 5.3.3 phần 4 để hiểu rõ hơn vấn đề này*). Việc trao đổi dữ liệu giữa chương trình và các thiết bị diễn ra thực chất đó là sự trao đổi giữa chương trình với các vùng đệm và các vùng đệm với các thiết bị thực sự. Ngôn ngữ lập trình C định nghĩa ra các *thiết bị logic* chuẩn sau: *stdin* (*thiết bị vào chuẩn – bàn phím*), *stdout* (*thiết bị ra chuẩn – màn hình*), *stderr* (*thiết bị lỗi chuẩn – màn hình*) và *stdprn* (*thiết bị in chuẩn – máy in*).

Thao tác *vào / ra* cơ bản được xét trong mục này là thao tác xuất ra màn hình và nhập vào từ bàn phím. Ngôn ngữ lập trình C cung cấp các hàm thư viện tương đối tiện dụng cho việc xuất nhập làm việc theo cả hai kiểu là *không* thông qua các dòng xuất nhập và thông qua các dòng xuất nhập.

1. Hàm printf

Là hàm ra chuẩn được khai báo trong tệp tiêu đề ***stdio.h*** của Turbo C dùng để trình bày dữ liệu ra màn hình theo một khuôn dạng nào đó do người viết chương trình định ra thông qua dòng xuất chuẩn *stdout*. Dạng tổng quát của hàm như sau:

```
int printf(const char *DieuKhien [,DanhSachCacDoi]);
```

Trong đó *DieuKhien* là một hằng con trỏ [7] kiểu *char* chứa địa chỉ của chuỗi điều khiển. Chuỗi điều khiển lại có thể bao gồm ba loại kí tự sau đây:

a) ***Các kí tự điều khiển***: Đó là các kí tự có mã từ *0* đến *31* và kí tự có mã *127* (kí tự *DEL*) trong bảng mã *ASCII*. Mỗi khi một kí tự điều khiển xuất hiện trong chuỗi điều khiển thì chức năng điều khiển của kí tự đó sẽ được thực hiện. Ví dụ nếu ta viết *printf("\n");* thì con trỏ màn hình sẽ được chuyển đến đầu dòng tiếp theo (*kí tự '\n' là kí tự đặc biệt cho trong bảng 1.4*).

b) ***Các kí tự hiển thị***: Đó là các kí tự còn lại trong bảng mã (*tuy nhiên các kí tự có mã từ 128 đến 255 là các kí tự đồ hoạ không có trên bàn phím*) sẽ được hiển thị ra màn hình mỗi khi hàm được gọi. Ví dụ để hiển thị chuỗi "*Hello !*" ra màn hình ta có thể viết lệnh *printf("Hello!");* . Ngoài các kí tự hiển thị thông thường, trong ngôn ngữ lập trình C có dùng một số kí tự như '\' , ''' (*dấu nháy đơn*), ',' (*dấu phẩy*), ' " ' (*dấu nháy kép*)... để biểu diễn cho các cú pháp riêng của ngôn ngữ. Do đó, để hiển thị được chúng ra màn hình chúng ta cần viết như sau:

```
printf(" \' "); sẽ in ra dấu '
printf(" \" "); sẽ in ra dấu "
printf(" \\ "); sẽ in ra dấu \
printf(" , "); sẽ in ra dấu ,
```

[7] Biến con trỏ sẽ được đề cập đến trong chương 3, mục 3.3.1.

c) ***Các đặc tả***: Dùng để đưa dữ liệu ra màn hình theo một khuôn dạng nhất định (dữ liệu này sẽ nằm trong *DanhSachCacDoi*). Mỗi đặc tả có cấu trúc tổng quát như sau:

%[-][fw][.pp] KyTuChuyenDang [8]

Trong đó dấu % là để chỉ ra rằng đây là bắt đầu của một *đặc tả* chứ không phải là kí tự hiển thị hay điều khiển. Trường '*fw*' là một số nguyên xác định độ rộng tối đa của trường ra. Khi *fw* lớn hơn độ dài thực tế của trường ra, thì các vị trí dư thừa sẽ được lấp đầy bởi các khoảng trống hoặc số 0 (*nếu số đầu tiên trong fw là 0*) và nội dung của trường ra sẽ được đẩy về bên phải (*nếu không có mặt dấu '-'*) hoặc về bên trái (*nếu có mặt dấu '-'*). Khi không có mặt *fw* hoặc khi *fw* nhỏ hơn hay bằng độ dài thực tế của trường ra thì độ rộng của trường ra sẽ bằng độ rộng thực tế.

'*pp*' là một số nguyên được sử dụng khi đối tương ứng là một xâu kí tự hoặc một giá trị kiểu *float* hay *double*. Nếu đối tương ứng là một số thực thì '*pp*' là độ chính xác sau dấu phẩy của trường ra. Khi vắng mặt '*pp*' trong trường hợp này thì độ chính xác được mặc định là 6. Khi đối tương ứng là một xâu kí tự thì chỉ có '*pp*' kí tự đầu tiên của xâu được hiển thị nếu '*pp*' nhỏ hơn độ dài của xâu.

Sau đây là một ví dụ minh hoạ cho những điều đã nói ở trên (*chú ý, độ dài trường ra được đặt trong cặp dấu ': :'*).

Ví dụ 1-11: Kết quả in ra màn hình tương ứng với các giá trị của *fw* và *pp*

Giá trị cần hiển thị	fw	dấu -	.pp	kết quả in ra
-2100	10	Có	Không	:-2100 :
-2100	010	Có	Không	:-2100 :
-2100	10	Không	Không	: -2100:
-2100	010	Không	Không	: 00000-2100:
"abcdef"	010	Có	Không	:abcdef :
"abcdef"	010	Không	Không	: abcdef:
"abcdef"	10	Không	Không	: abcdef:
"abcdef"	3	Có	Không	:abcdef:
"abcdef"	3	Không	Không	:abcdef:
123456	4	Có	Không	:123456:
123456	4	Không	Không	:123456:
-123.456	10	Có	2	:-123.46 :
-123.456	10	Có	0	:-123 :
-123.456	8	Có	Không	:-123.456000:
-123.456	10	Không	2	: -123.46:
"abcdefghi"	10	Có	3	:abc :
"abcdefghi"	10	Không	6	: abcdef:
"abcdefghi"	Không	Không	Không	:abcdefghi:
"abcdefghi"	10	Không	Không	: abcdefghi:

KyTuChuyenDang là một hoặc một dãy kí hiệu, nó xác định quy tắc chuyển dạng của giá trị cần hiển thị. Mỗi *kiểu dữ liệu* đều có một *kí tự chuyển dạng* cho riêng nó. Các kí tự chuyển dạng trong ngôn ngữ lập trình C được cho trong bảng dưới đây.

---

[8] Quy ước rằng, tất cả những gì viết trong cặp dấu [] thì có thể có mặt hoặc vắng mặt.

Bảng 1.5. Các kí tự chuyển dạng cho hàm *printf*

Kí tự chuyển dạng	Kiểu của giá trị cần hiển thị	Cách chuyển dạng
d hoặc i	int	Giá trị cần hiển thị được coi là số nguyên hệ 10 có dấu.
ld hoặc li	long	Giá trị cần hiển thị được coi là số nguyên dài hệ 10 có dấu.
o	int	Giá trị cần hiển thị được coi là số nguyên hệ 8 không dấu.
lo	long	Giá trị cần hiển thị được coi là số nguyên dài hệ 8 không dấu.
u	int	Giá trị cần hiển thị được coi là số nguyên hệ 10 không dấu.
X hoặc x	int	Giá trị cần hiển thị được coi là số nguyên hệ 16 không dấu (nếu là X thì sẽ được hiện ra dưới dạng chữ hoa).
lx	long	Giá trị cần hiển thị được coi là số nguyên dài hệ 16 không dấu.
e, E	float, double	Giá trị cần hiển thị được coi là số thực và được hiển thị dưới dạng kí pháp khoa học (dạng mũ) với độ dài phần thập phân có giá trị là pp.
f	float, double	Giá trị cần hiển thị được coi là số thực và được hiển thị dưới dạng số thập phân dấu phẩy tĩnh.
g,G	float, double	Giá trị cần hiển thị được chuyển sang dạng số thập phân dấu phẩy tĩnh hay động tùy thuộc vào loại nào ngắn hơn. Không in ra các số không vô nghĩa.
c	char	Giá trị cần hiển thị được coi là một kí tự.
s	char *	Giá trị cần hiển thị được coi là một xâu kí tự.

DanhSachCacDoi có thể chứa các *giá trị* (*các hằng*), các *biến* hay các *biểu thức* [9] muốn hiển thị ra màn hình. Mỗi *đối số* (*giá trị, biến hoặc biểu thức*) được phân tách nhau bởi một dấu phẩy. Mỗi khi thực hiện lệnh, giá trị của *đối số* tương ứng [10] (*tính theo thứ tự từ trái qua phải*) sẽ được *chuyển dạng* và hiển thị ra màn hình theo *đặc tả tương ứng* (*cũng tính theo thứ tự từ trái qua phải*). Những đối nào không có đặc tả tương ứng sẽ không được in ra.

Ví dụ 1-12. Đưa dữ liệu ra màn hình có sử dụng đặc tả.

```
int a=10, b=20;
float c=1.2;
printf("Cac gia tri la:\nA=%d\nB=%d",a,b,c);
```

Nếu các câu lệnh trên được thực hiện sẽ cho kết quả như sau:

```
Cac gia tri la:
A=10
B=20
```

Biến c không có đặc tả cho nên không được in ra.

Sau đây là một ví dụ minh hoạ cho cách dùng hàm *printf()*.

Ví dụ 1-13. Lập trình in ra màn hình một danh sách lớp gồm *4* cột lần lượt là *Ho va Ten*, *Ngay Sinh*, *So hieu SV* và *Diem TBT* sao cho cột *Ho va Ten* có độ rộng là *25* kí tự, các tên căn theo lề trái. Cột *Ngay Sinh* có độ rộng là *15* kí tự, dữ liệu căn theo lề phải. Cột *So hieu SV* có độ rộng là *15* kí tự, dữ liệu căn theo lề phải. Cột

---

[9] Biểu thức sẽ được đề cập đến trong chương 2.

[10] Nếu đối số là một biểu thức thì giá trị của biểu thức sẽ được tính trước, sau đó giá trị này mới được chuyển dạng theo đặc tả tương ứng và hiển thị ra màn hình.

Diem TBT có độ rộng là *15* kí tự, dữ liệu căn theo lề phải và có độ chính xác là *2* số sau dấu thập phân.

```
/* ************************************************************************************ */
#include "stdio.h"
#include "conio.h"
/* ************************************************************************************ */
int main()
{
  printf("\n\n\n\t\t\tDANH SACH SINH VIEN\n\n\n");
  printf("%-25s%15s%15s%15s\n\n","Ho va Ten" , "Ngay Sinh" , "So hieu SV" ,
  "Diem TBT");
  printf("\n%-25s%15d%15s%15.2f " , "Nguyen Van A",11,"BK2003  A120", 6.246);
  printf("\n%-25s%15d%15s%15.2f " , "Nguyen Van B",13,"BK2003  A121", 7.146);
  printf("\n%-25s%15d%15s%15.2f " , "Nguyen Van C",21,"BK2003  A122", 8.2);
  printf("\n%-25s%15d%15s%15.2f " , "Nguyen Van D",18,"BK2003  A123", 5.125);
  printf("\n%-25s%15d%15s%15.2f " , "Nguyen Van E",15,"BK2003  A124", 4.925);
  getch();
  return 0;
}
/* ************************************************************************************ */
```

Kết quả chạy chương trình:

```
                        DANH SACH SINH VIEN

Ho va Ten                Ngay Sinh     So hieu SV       Diem TBT

Nguyen Van A                    11     BK2003A120           6.25
Nguyen Van B                    13     BK2003A121           7.15
Nguyen Van C                    21     BK2003A122           8.20
Nguyen Van D                    18     BK2003A123           5.13
Nguyen Van E                    15     BK2003A124           4.93
/* ************************************************************************************ */
```

Chú ý:

- Nếu một đặc tả viết không đúng cấu trúc (*nghĩa là không bắt đầu bằng kí tự % hoặc không kết thúc bằng một kí tự chuyển dạng*) thì tất cả các kí tự đó sẽ được coi là kí tự hiển thị và sẽ được hiện ra màn hình. Ví dụ *printf("x=%h",x);* thì kết quả chạy sẽ là: *x=%h.*

- Giữa chuỗi điều khiển và danh sách các đối số phải được phân tách nhau bởi một dấu phẩy. Sau đây là một ví dụ sai về hàm *printf()*:

```
printf("Toi noi rang\n%-20s" "Hello World");
```

Vì "*Hello World*" là một đối số cần hiển thị do đó nó phải được phân biệt với chuỗi điều khiển "Toi noi rang\n%-20s" bởi một dấu phẩy, ta viết lại như sau:

```
printf("Toi noi rang\n%-20s","Hello World");
```

- Để hiển thị ra màn hình các kí tự đồ hoạ (*không có mặt trên bàn phím*) thì ta phải sử dụng mã *ASCII* của các kí tự đó trong hàm *printf()*. Ví dụ, để hiển thị kí tự **'Ç'** ra màn hình ta phải viết: *printf("%c", 128);* giá trị *128* chính là mã *ASCII* của kí tự **'Ç'** trong bảng mã.
- Nếu sử dụng sai kí tự chuyển dạng thì sẽ dẫn đến kết quả hiển thị sai.
- Mỗi đối số cần in ra phải có một đặc tả riêng cho nó.
- Hàm *printf()* sẽ trả về số kí tự được đưa ra màn hình nếu thành công (*bao gồm cả kí tự điều khiển*), khi có lỗi hàm trả về giá trị -1.

Ví dụ 1-14. Giá trị trả về của hàm *printf*. Xét đoạn chương trình sau:

```
int a=30, b=3265, m;
m=printf("\nA=%4d B=%d", a, b);
```

Thì *m* sẽ có giá trị là 14.

2. Hàm scanf

Là hàm vào chuẩn được khai báo trong thư viện ***stdio.h*** dùng để nhập dữ liệu từ bàn phím theo một khuôn dạng xác định thông qua dòng nhập chuẩn *stdin* rồi lưu trữ các giá trị đã chuyển đổi đó vào bộ nhớ tại các địa chỉ tương ứng trong *DanhSachDoi*. Dạng tổng quát của hàm như sau:

```
int scanf(const char *DieuKhien  [, DanhSachDoi]);
```

Trong đó *DieuKhien* là một hằng con trỏ kiểu *char* chứa địa chỉ của chuỗi điều khiển. Còn *DanhSachDoi* là danh sách địa chỉ của các biến dùng để lưu trữ giá trị đọc vào từ bàn phím. Các *đối số* (*các giá trị địa chỉ của các biến*) cần được phân tách nhau bởi một dấu phẩy. Để lấy được địa chỉ của một biến ta dùng toán tử '*&*' đứng trước tên biến đó. Ví dụ để lấy địa chỉ của biến *x* ta viết *&x*.

Chuỗi điều khiển của hàm *scanf()* thường chỉ bao gồm ***các đặc tả*** cho việc chuyển dạng dữ liệu (*không có kí tự điều khiển hay kí tự hiển thị*). Mỗi đặc tả sẽ được tương ứng với một địa chỉ của biến, xác định từ trái qua phải. Biến nào không có đặc tả cho nó thì sẽ không được nhập giá trị. Mỗi đặc tả cho hàm *scanf()* có thể được mô tả tổng quát như sau:

```
%[*][dd]KyTuChuyenDang
```

Trong đó kí tự *%* là dấu hiệu để chỉ ra sự bắt đầu của một đặc tả. Kí tự '*' khi có mặt sẽ cho phép ***dòng vào*** vẫn được dò đọc bình thường nhưng giá trị tương ứng đọc được bị bỏ qua (*không được lưu vào các biến nhớ*) do đó sẽ không có các đối tương ứng (*xem ví dụ 1-16*).

Dòng vào ở đây được hiểu là một dãy kí tự liên tiếp nhập từ bàn phím đã được chuyển tới *stdin* và kết thúc bằng phím *Enter*. Mỗi dòng vào bao gồm một hay nhiều ***trường vào*** đó là một dãy kí tự liên tiếp nhau không chứa dấu cách, dấu Tab hoặc kí tự xuống dòng. Các trường vào có thể được phân tách nhau bởi một hay nhiều dấu cách, dấu Tab hay dấu xuống dòng. Mỗi trường vào sẽ được xét tương ứng cho một đối số trong *DanhSachDoi* (*theo thứ tự từ trái qua phải*). Ví dụ xét dòng vào sau đây:

```
1234 abc    xyz
456  Ghy  (Enter)
```

Sẽ gồm có *5* trường vào đó là *1234*, *abc*, *xyz*, *456* và *Ghy*.

Trường *dd* trong đặc tả là một giá trị nguyên xác định chiều dài cực đại của *trường vào* sẽ được đọc cho đối tương ứng. Nếu *dd* vắng mặt hoặc nếu giá trị của nó lớn hơn hay bằng độ dài của trường vào tương ứng thì toàn bộ trường vào đó sẽ được đọc và *chuyển dạng* theo đặc tả tương ứng cho nó, sau đó giá trị này mới được lưu vào biến tương ứng có địa chỉ trong *DanhSachDoi* (*nếu không có dấu * được chỉ định*). Nếu giá trị của *dd* nhỏ hơn độ dài của trường vào tương ứng thì chỉ phần đầu của trường vào có kích cỡ đúng bằng *dd* được đọc và *chuyển dạng* sau đó lưu vào biến tương ứng. Phần còn lại của trường vào sẽ được gán cho các đối tiếp theo.

KyTuChuyenDang sẽ xác định cách thức dò đọc các kí tự trên dòng vào cũng như phương pháp chuyển dạng thông tin đọc được trước khi gán nó cho các biến tương ứng. Danh sách các kí tự chuyển dạng được cho trong bảng sau:

Bảng 1.6. Các kí tự chuyển dạng cho hàm *scanf*

Ký tự đặc tả	Mô tả chức năng
i, d	Nhập vào một số nguyên; đối số tương ứng phải là con trỏ nguyên hoặc địa chỉ của một biến nguyên. Trường vào phải là một số nguyên.
li, ld	Nhập vào một số nguyên dài; đối số tương ứng phải là con trỏ kiểu long hoặc là địa chỉ của một biến long. Trường vào phải là một số nguyên.
o	Nhập vào một số nguyên hệ 8; đối số tương ứng phải là con trỏ kiểu nguyên hoặc là địa chỉ của một biến nguyên. Trường vào phải là một số nguyên hệ 8.
lo	Nhập vào một số nguyên dài hệ 8; đối số tương ứng phải là con trỏ kiểu long hoặc là địa chỉ của một biến long. Trường vào phải là một số nguyên hệ 8.
x	Nhập vào một số nguyên hệ 16; đối số tương ứng phải là con trỏ kiểu nguyên hoặc là địa chỉ của một biến nguyên. Trường vào phải là một số nguyên hệ 16.
lx	Nhập vào một số nguyên dài hệ 16; đối số tương ứng phải là con trỏ kiểu long hoặc là địa chỉ của một biến long. Trường vào phải là một số nguyên hệ 16.
c	Nhập vào một kí tự; đối số tương ứng phải là một con trỏ kí tự hoặc địa chỉ của biến kí tự. Trường vào là một kí tự bất kì.
s	Nhập vào một xâu kí tự; đối số tương ứng phải là con trỏ kiểu kí tự hoặc là địa chỉ của một mảng kí tự. Trường vào là một dãy kí tự bất kì không chứa dấu cách, Tab và dấu xuống dòng. Kí tự NULL '\0' sẽ được tự động thêm vào cuối của kết quả nhận được.
f hoặc e	Nhập vào một số thực kiểu float; đối số tương ứng phải là con trỏ kiểu float hoặc là địa chỉ của một biến float. Trường vào phải là một số thực.
lf hoặc le	Nhập vào một số thực kiểu double; đối số tương ứng phải là con trỏ kiểu double hoặc là địa chỉ của một biến double. Trường vào phải là một số thực.

Ví dụ 1-15. Cách dùng đặc tả trong hàm *scanf*. Xét đoạn chương trình sau:

```
int k ;
float x, y ;
char ch1[6], ch2[6] ;
```

```
scanf("%f%5f%3d%3s%s", &x, &y, &k, ch1, ch2) (11);
```

Thì với dòng vào như sau:

```
34.24e-1 36 12345678b (Enter)
```

hàm sẽ biến đổi các kí tự của toàn bộ trường vào thứ nhất "*34.24e-1*" thành dạng số thực dấu phẩy tĩnh *3.424* và gán cho biến *x*; các kí tự của toàn bộ trường vào thứ hai "*36*" thành dạng số thực dấu phẩy tĩnh *36.0* và gán cho biến *y*; ba kí tự đầu tiên của trường vào thứ ba "*123*" thành dạng số nguyên *123* và gán cho biến *k*; ba kí tự tiếp theo của trường vào thứ ba "*456*" thành dạng xâu kí tự "*456*" và gán cho biến *ch1*; và các kí tự còn lại của trường vào thứ ba "*78b*" thành dạng xâu kí tự "*78b*" và gán cho biến *ch2* (*lưu ý ch1 và ch2 đều đã bao gồm kí tự kết thúc chuỗi \0*).

Ví dụ 1-16. Chương trình minh hoạ cách dùng đặc tả %* trong *scanf()*.

```
/* ******************************************************************************** */
#include"stdio.h"
#include"conio.h"
/* ******************************************************************************** */
int main()
{
    int a;
    long b;
    float c, d;
    char Xau[25], ch;
    printf("\n Hay nhap cac gia tri tuong ung cho a, b, c, d, Xau, va ch: \n"); (12)
    scanf("%3d%ld%f%f%s%*c%c", &a, &b, &c, &d, Xau, &ch);
    printf("\n Gia tri cua cac bien sau khi nhap la:\n");
    printf("a=%10d\n", a);
    printf("b=%10ld\n", b);
    printf("c=%10.2f\n", c);
    printf("d=%10.2f\n", d);
    printf("Xau=\"%10s\"\nch='%10c'\n", Xau, ch);
    getch();
    return 0;

}
/* ******************************************************************************** */
```

Kết quả chạy chương trình:

```
Hay nhap cac gia tri tuong ung cho a, b, c, d, Xau va ch:
12345678  254.45e-3  573  ABC 67890
Gia tri cua cac bien sau khi nhap la:
a=         123
b=       45678
c=        0.25
d=      573.00
```

---

[11] Hai biến **ch1** và **ch2** là các mảng kí tự cho nên ta không phải dùng toán tử xác định địa chỉ '&' cho chúng. Trong chương 3, mục 3.3.2 sẽ bàn kĩ hơn về vấn đề này.

[12] Trong Turbo C 2.0 ta có thể hiển thị trực tiếp các kí tự ',' và kí tự ''' mà không cần sử dụng '\,' và '\'' trong chuỗi điều khiển.

```
Xau="          ABC"
ch='           6'
/* ****************************************************************************** */
```

Trong ví dụ trên, nếu ta bỏ đi đặc tả *%*c* trong hàm *scanf()* thì kết quả sẽ không hiện được dòng *ch*='   *6*'. Nguyên nhân là do sau khi nhập xâu kí tự xong, máy sẽ đọc kí tự tiếp theo (*mà không phân biệt dấu cách hay dấu xuống dòng do dùng đặc tả %c ở sau*) vào biến *ch*. Do đó thực chất biến *ch* sẽ chứa dấu cách ' ' (*nếu ta nhập trên một dòng*) hoặc chứa dấu xuống dòng '*\n*' (*nếu ta nhập trên nhiều dòng*) và số 6 không được in ra. Để xử lí tình huống này, ta thêm một đặc tả *%*c* vào trước đặc tả *%c* khi đó đặc tả *%*c* sẽ vẫn đọc dấu cách *hoặc* dấu xuống dòng như một kí tự bình thường nhưng không lưu vào biến nào cả.

Chú ý:

- Cũng giống như hàm *printf()*, giữa chuỗi điều khiển và danh sách các đối phải được phân tách với nhau bằng một dấu phẩy.

- Không nên sử dụng trường *d* trong đặc tả nếu không biết rõ độ dài trường vào. Đồng thời cần lưu ý số *đối số*, số các *đặc tả* và số *trường vào* phải bằng nhau.

- Khi nhập cho nhiều đối cùng một lúc thì các giá trị tương ứng với các đối trên dòng vào có thể được viết trên cùng một dòng (*nhưng các trường vào phân tách nhau bởi một hoặc nhiều dấu cách hoặc Tab*) hoặc trên nhiều dòng khác nhau.

- Không thể nhập một chuỗi gồm *nhiều từ* cho biến chuỗi bằng lệnh ***scanf***.

Ví dụ. Xét đoạn chương trình sau:

```
char Ten[]; /*Khai báo một mảng kí tự*/
scanf("%s", Ten);
```

Thì khi ta gõ vào dòng sau đây: Nguyen Thi Lan Anh (*Enter*)

Biến ***Ten*** sẽ chỉ chứa được chuỗi *Nguyen* mà thôi. Để có thể nhập đúng được cả họ và tên ta cần dùng hàm nhập ***gets*** được khai báo trong ***stdio.h*** thay thế cho hàm ***scanf*** (*xem thêm mục 3.3.2*).

- Chương trình sẽ thoát khỏi hàm *scanf()* khi một trong các điều kiện sau đây xảy ra:

+ hoặc đã xét hết các đặc tả (cho dù vẫn còn đối)
+ hoặc đã xét hết các đối (mặc dù vẫn còn đặc tả)
+ hoặc gặp một sự không tương thích giữa các đặc tả và đối tương ứng (nhập sai kiểu).

Khi trở về chương trình, hàm *scanf()* sẽ trả về một giá trị nguyên bằng các trường vào đã được chuyển dạng và lưu vào bộ nhớ.

- Khi thoát khỏi hàm *scanf* kí tự '*\n*' (*Enter*) vẫn còn nằm lại trong *stdin* và có thể làm ảnh hưởng đến các câu lệnh khác cũng đang làm việc với *stdin* (*ví dụ làm trôi hàm gets, getchar... trong mục 4 dưới đây*). Để khử kí tự '*\n*' trong *stdin* người ta dùng một trong những cách sau:

*+) Thêm đặc tả %*c vào cuối chuỗi điều khiển của mỗi hàm scanf.*
*+) Dùng hàm **fflush(stdin)** để làm sạch stdin.*

- Dòng vào sẽ được dò đọc theo từng kí tự (*kể cả dấu cách, Tab và kí tự xuống dòng*) nếu ta sử dụng các kiểu chuyển dạng đặc biệt sau đây: *c*, *[...]* và *[^...]*. Với kiểu chuyển dạng *[tập hợp các kí tự nào đó]* hàm *scanf()* sẽ lần lượt đọc các kí tự trên dòng vào cho đến khi gặp một kí tự không thuộc tập các kí tự đã được chỉ định trong cặp dấu []. Đối tương ứng phải là một *con trỏ kiểu char* trỏ tới một vùng nhớ đủ lớn. Trường vào là một dãy kí tự bất kì.

Với kiểu chuyển dạng *[^ tập các kí tự nào đó]* hàm *scanf()* sẽ lần lượt đọc các kí tự trên dòng vào cho đến khi gặp một kí tự thuộc tập các kí tự đã được chỉ định trong cặp dấu []. Đối tương ứng phải là một con trỏ kiểu char. Trường vào là một dãy kí tự bất kì.

Ví dụ 1-17. Chương trình minh hoạ cách sử dụng của hai kiểu chuyển dạng đặc biệt *[...]* và *[^...]*.

```
/* ******************************************************************************** */
#include"stdio.h"
#include"conio.h"
/* ******************************************************************************** */
int main()
{
    int a, b;
    char Xau1[25], Xau2[25];
    printf("\n Hay nhap cac gia tri tuong ung cho a, Xau1, Xau2 va b:\ \n");
    scanf("%d%*c%[0123456789]%[^0123456789]%3d",
    &a, Xau1, Xau2, &b);
    printf("\n Gia tri cua cac bien sau khi nhap la:\n");
    printf("a=%d\n", a);
    printf("Xau1=\"%s\"\nXau2=\"%s\"\n", Xau1, Xau2);
    printf("b=%d\n", b);
    getch();
    return 0;
}
/* ******************************************************************************** */
```

Kết quả chạy chương trình:

```
Hay nhap cac gia tri tuong ung cho a, Xau1, Xau2 va b:
35 12345  ABC67890
Gia tri cua cac bien sau khi nhap la:
a=35
Xau1="12345"
Xau2="  ABC"
b=678
/* ******************************************************************************** */
```

Bài tập:

a) Hãy giải thích hoạt động của chương trình trên.

b) Tại sao phải sử dụng đặc tả *%*c* trong câu lệnh *scanf* ở trên, nếu không có đặc tả này thì kết quả in ra màn hình sẽ như thế nào?

c) Nếu trong dòng vào, khoảng cách giữa số *35* và *12345* là hai dấu cách thì hiện tượng gì xảy ra?

d) Tại sao trong *Xau2* các kí tự đầu lại chứa hai dấu cách?

3. Các hàm khác dùng để trình bày dữ liệu ra màn hình

a) ***Hàm puts*** (*được khai báo trong stdio.h*)

Dùng để đưa một chuỗi kí tự ra màn hình thông qua *stdout*.

Dạng hàm:

```
int puts(const char *s);
```

Hoạt động:

Hàm sẽ đưa chuỗi do con trỏ *s* quản lí và một kí tự '\n' lên *stdout*. Nếu thành công, hàm trả về kí tự cuối cùng được xuất (*chính là kí tự '\n'*), ngược lại hàm trả về EOF [13].

Ví dụ câu lệnh puts("Hello"); sẽ đưa ra màn hình dòng chữ "Hello" sau đó xuống dòng (*tương đương với câu lệnh printf("Hello\n");*).

b) ***Hàm putchar*** (*được khai báo trong stdio.h*)

Dùng để đưa một kí tự ra màn hình thông qua *stdout*.

Dạng hàm:

```
int putchar(int c);
```

Hoạt động:

Hàm sẽ đưa một kí tự *c* dưới dạng mã *ASCII* ra *stdout*, nếu thành công hàm trả về kí tự được xuất, ngược lại hàm trả về giá trị EOF.

Ví dụ câu lệnh putchar ('B'); sẽ đưa mã *66* ra *stdout* mà kết quả là một kí tự '*B*' được hiển thị ra màn hình tại vị trí hiện thời của con trỏ màn hình.

c) ***Hàm putch*** (*khai báo trong conio.h*)

Dùng để đưa một kí tự ra cửa sổ văn bản màn hình (*không qua stdout*).

Dạng hàm:

```
int putch(int c);
```

Hoạt động:

Đưa một kí tự có mã *ASCII* là *c* lên cửa sổ văn bản màn hình. Kí tự sẽ được hiển thị theo màu xác định trong hàm *textcolor*.

Ví dụ các câu lệnh sau textcolor(RED); putch('A'); sẽ cho hiện một chữ 'A' màu đỏ lên màn hình.

4. Các hàm khác dùng để nhập dữ liệu vào từ bàn phím

a) ***Hàm gets*** (*được khai báo trong stdio.h*)

Dùng để nhập một chuỗi kí tự từ bàn phím thông qua *stdin*.

Dạng hàm:

```
char * gets(char *s);
```

Hoạt động:

Hàm tiến hành nhận một dãy kí tự từ *stdin* cho đến khi gặp kí tự '\n' (*do đó nếu trong stdin đã có sẵn kí tự '\n' rồi thì hàm gets sẽ không chờ người sử dụng*

---

[13] EOF là một tên hằng được định nghĩa trong stdio.h. Hãy xem thêm chương 5.

nhập dữ liệu vào nữa, ta nói hàm gets đã bị trôi). Kí tự '*\n*' sẽ bị loại khỏi *stdin* nhưng không được đặt vào chuỗi. Chuỗi nhận được sẽ được tự động bổ sung thêm kí tự '*\0*' để đánh dấu sự kết thúc chuỗi rồi được đặt vào vùng nhớ do con trỏ *s* trỏ tới. Hàm trả về địa chỉ của chuỗi nhận được.

Ví dụ để nhập từ bàn phím một chuỗi kí tự rồi lưu vào biến *HoTen* ta viết như sau: char HoTen[25]; gets(HoTen);

b) ***Hàm getchar*** (*được khai báo trong stdio.h*)

Dùng để nhận một kí tự từ *stdin*.

Dạng hàm:

```
int getchar(void);
```

Hoạt động:

Hàm sẽ tiến hành nhận mã *ASCII* của một kí tự từ *stdin* và trả về kí tự vừa nhận được.

Ví dụ để nhập một kí tự từ bàn phím vào biến *c* ta có thể viết như sau:
char c; c=getchar();

c) ***Hàm getch và getche*** (*khai báo trong conio.h*)

Dùng để nhận một kí tự trực tiếp từ bộ đệm bàn phím (*không qua stdin*).

Dạng hàm:

```
int getch(void);
int getche(void);
```

Hoạt động:

Nếu trong bộ đệm bàn phím có sẵn kí tự, thì hàm nhận một kí tự trong đó. Nếu bộ đệm rỗng thì máy tạm dừng cho đến khi người sử dụng gõ vào một phím bất kì (*không bắt buộc phải kết thúc bằng phím Enter như các hàm làm việc với stdin do đó các hàm này thường được sử dụng để bắt phím khi viết Menu...*). Phím vừa gõ sẽ được hiện lên màn hình nếu đang dùng hàm *getche*, còn với hàm *getch* thì kí tự không được hiện ra màn hình. Hàm sẽ trả về kí tự vừa gõ.

5. Đặc điểm của các hàm làm việc với stdin

- Nếu trên *stdin* đã có sẵn dữ liệu thì các hàm này sẽ nhận một phần dữ liệu đó tuỳ theo yêu cầu của từng hàm. Phần dữ liệu còn lại (*nếu có*) vẫn ở trên *stdin*.
- Phím *Enter* dùng để kết thúc quá trình nhập sẽ vẫn nằm lại trên *stdin*.
- Nếu *stdin* không có hoặc không đủ dữ liệu theo yêu cầu của các hàm thì máy sẽ tạm dừng để chờ người sử dụng đưa thêm dữ liệu từ bàn phím lên *stdin* cho đến khi phím *Enter* được nhấn.

Ví dụ 1- 18. Xét đoạn chương trình sau:

```
...
char ch1, ch2;
int a, b;
printf("\nHay nhap gia tri cho cac bien ch1, a, b va ch2");
ch1=getchar(); /* hàm làm việc với stdin */
scanf("%d%d", &a, &b); /* hàm làm việc với stdin */
ch2=getche(); /* hàm làm việc trực tiếp với bộ đệm bàn phím */
```

```
printf("\nCac gia tri ch1= %c, a= %d, b= %d va ch2= %c");
...
```

Nếu đoạn chương trình trên được thực hiện thì với dòng vào:

```
A123  456B (Enter)
V
```

sẽ cho kết quả như sau: Cac gia tri ch1= A, a= 123, b= 456 va ch2= V

Có được kết quả này là do khi gặp hàm *getchar* tất cả các kí tự nhập từ bàn phím sẽ được chuyển tới *stdin* cho đến khi gặp kí tự '*\n*'. Kí tự đầu tiên của dòng vào sẽ được chuyển cho hàm *getchar* và gán cho biến *ch1* (*kí tự A*). Các kí tự còn lại vẫn còn nằm trên *stdin* (*bao gồm 123 456B và kí tự '\n'*). Do đó khi gặp câu lệnh tiếp theo scanf("%d%d", &a, &b); máy không cần chờ người sử dụng nhập dữ liệu vào từ bàn phím nữa mà chuyển trường vào thứ nhất "*123*" có sẵn trong *stdin* thành số nguyên theo đặc tả và đưa vào biến *a*; trường vào thứ hai "*456B*" thành số nguyên *456* (*kí tự 'B' vẫn còn trên stdin*) và đưa vào biến *b*. Khi gặp hàm *getche* do bộ đệm bàn phím vẫn chưa có kí tự nào (*mặc dù stdin vẫn còn kí tự 'B'*) cho nên máy sẽ chờ người sử dụng bấm một phím bất kì, phím này sẽ được chuyển cho hàm *getche* và đưa vào biến *ch2* (*kí tự 'V'*).

1.3. CẤU TRÚC MỘT CHƯƠNG TRÌNH C ĐƠN GIẢN

Cấu trúc đơn giản của một chương trình C thường được viết theo trình tự sau:

+ Các chỉ thị tiền xử lí

    #include

    #define

+ Hàm ***main***

Trong đó, các chỉ thị tiền xử lí dùng để hiệu chỉnh lại chương trình nguồn trước khi biên dịch nó thành một chương trình chạy được. Trong ngôn ngữ lập trình C, có nhiều chỉ thị tiền xử lí được sử dụng với nhiều mục đích khác nhau, ý nghĩa và cách sử dụng của từng chỉ thị cụ thể được trình bày thêm trong phụ lục II. Trong chương này, ta chỉ quan tâm chủ yếu tới chỉ thị *#include* và chỉ thị *#define*. Chỉ thị *#define* chúng ta đã làm quen trong mục 1.2.7. Chỉ thị này dùng để định nghĩa ra các *Tên hằng* và các *Macro* có ***phạm vi toàn cục*** từ lúc định nghĩa cho đến cuối tệp gốc hoặc cho đến khi được định nghĩa lại sau chỉ thị *#undef*. Có nghĩa là một *Tên hằng* hay một *Macro* từ khi được định nghĩa ra bởi chỉ thị *#define* thì giá trị của nó luôn được thay thế ở bất kì đâu trong chương trình mà nó xuất hiện (*không phụ thuộc vào việc nó nằm trong khối lệnh hay ở bên ngoài các khối lệnh*).

Chỉ thị bao hàm tệp *#include* <[Đường_dẫn\]tên_tệp> dùng để chèn nội dung của một tệp khác vào chương trình nguồn tại đúng vị trí mà chỉ thị đó xuất hiện. Trước khi dịch, chương trình biên dịch Turbo C sẽ tìm tệp theo tên và đường dẫn ghi trong <Đường_dẫn>, nếu tìm thấy thì toàn bộ tệp đó sẽ được chèn vào chương trình nguồn tại nơi chỉ thị *#include* xuất hiện, ngược lại một thông báo lỗi sẽ được đưa ra:

Unable to open include file '[Đường_dẫn] tên tệp'

Có hai cách viết khác nhau cho chỉ thị *#include*. Đó là:

```
#include <[Đường_dẫn\]tên_tệp>
#include "[Đường_dẫn\]tên_tệp"
```

Cách thức làm việc của hai dạng này chỉ khác nhau khi không có thông tin về đường dẫn. *Theo dạng 1*, trình biên dịch sẽ chỉ tìm tệp cần chèn trong thư mục INCLUDE của Turbo C. *Còn theo dạng 2*, trước tiên trình biên dịch sẽ tìm tệp cần chèn trong thư mục chủ, nếu không thấy thì tiếp tục tìm trong thư mục INCLUDE của Turbo C.

Chú ý:

- Chỉ thị *#include* phải được viết trên một dòng.
- Chỉ thị *#include* có thể bao hàm nhau trong các tệp.

Ví dụ, nếu trong tệp gốc *CTChinh.c* có chứa chỉ thị *#include "a1.c"*. Trong tệp *a1.c* này đến lượt nó lại chứa chỉ thị *#include "a2.c"*. Khi biên dịch tệp *CTChinh.c* thì điều gì sẽ xảy ra? Trước tiên, trình biên dịch sẽ tìm tệp *a2.c* trong thư mục chủ và trong thư mục INCLUDE, nếu tìm thấy nội dung tệp này sẽ được chèn vào đúng vị trí của chỉ thị *#include "a2.c"* trong tệp *a1.c* (*nếu tệp này tồn tại hoặc trong thư mục INCLUDE hoặc trong thư mục chủ*). Ta nhận được tệp *a1.c* mới, tệp mới này lại được chèn thay thế cho chỉ thị *#include "a1.c"* trong tệp gốc. Như vậy trước khi biên dịch cả hai tệp *a1.c* và *a2.c* đều được ghép vào tệp chương trình gốc nhờ hai chỉ thị *#include*.

- Sử dụng chỉ thị *#include* ta có thể tổ chức một chương trình lớn trên nhiều tệp khác nhau do nhiều nhóm làm việc khác nhau mà về mặt logic vẫn được coi là viết trên một tệp.

- Dùng chỉ thị *#include* tạo khả năng cho người lập trình tự xây dựng ra các thư viện của riêng mình. Tuy nhiên, việc tổ chức các tệp thư viện và các mô đun chương trình phải tuân thủ một số quy tắc nhất định, điều này sẽ được trình bày kĩ hơn trong *phụ lục II*.

- Chỉ thị *#include* cũng thường được sử dụng trong ngôn ngữ lập trình C để ghép các tệp tiêu đề có dạng **.h* vào chương trình. Các tệp tiêu đề của Turbo C (*như stdio.h, conio.h, dos.h...*) dùng để chứa khai báo của các hàm chuẩn, định nghĩa các hằng và các kiểu dữ liệu chuẩn trong ngôn ngữ. Trong một chương trình ứng dụng, nếu ta muốn sử dụng lại các hàm, các hằng hay các kiểu dữ liệu chuẩn đã được định nghĩa này, ta phải dùng chỉ thị *#include* để kết nối một tệp tiêu đề tương ứng có chứa khai báo đó. Ví dụ, để sử dụng các hàm *vào / ra* trong Turbo C ta cần thêm vào chương trình gốc câu lệnh sau: *#include "stdio.h"*.

Hàm *main* lại có thể là một trong những dạng sau đây:

Dạng 1:

```
main()
{
	/* Các khai báo nằm ở đây*/
	/* Các lệnh nằm ở đây*/
}
```

Dạng 2:

```
Kiểu_trả_về  main(void)
{
	/* Các khai báo nằm ở đây*/
```

```
        /* Các lệnh nằm ở đây*/
        return Giá_trị_trả_về;
}
```

Dạng 3:

```
[Kiểu_trả_về] main(int i, char *str[])
{
        /* Các khai báo nằm ở đây*/
        /*Các lệnh nằm ở đây*/
        return Giá_trị_trả_về;
}
```

Mỗi hàm ***main*** nhìn chung bao gồm 2 phần, phần đầu hàm và phần thân hàm. Phần đầu hàm bao gồm *tên hàm* và có thể có đối số cũng như giá trị trả về. Phần thân hàm bắt đầu bằng một kí tự '{' và kết thúc bằng một kí tự '}'. Bên trong thân hàm là toàn bộ các lệnh được sử dụng trong chương trình. ***Dạng 1*** và ***Dạng 2*** là hai cấu trúc thường được sử dụng nhất, chúng chỉ phân biệt nhau ở giá trị trả về của hàm. Giá trị trả về là giá trị theo một kiểu nào đó mà hệ điều hành nhận được sau khi chương trình kết thúc. Hệ điều hành sẽ căn cứ vào giá trị trả về của mỗi chương trình để nhận biết tình trạng hoạt động của các chương trình đó.

Dạng 3 là cấu trúc đối dòng lệnh, nó cho phép một chương trình khi chạy có thể có tham số đầu vào (*ví dụ setup /?...*). Trong đó, *i* sẽ chứa số tham số, còn các phần tử của mảng hình thức *str[]* sẽ chứa địa chỉ của các tham số đưa vào dưới dạng các chuỗi kí tự, mỗi tham số sẽ phân biệt nhau bởi ít nhất một dấu cách. Tham số đầu tiên sẽ là tên chương trình cùng với đường dẫn và được lưu trong *str[0]*, các tham số còn lại sẽ là những tham số của chương trình. Trong thân hàm *main* chúng ta có thể sử dụng *i* và *str* như các tham số đầu vào được nhập từ bàn phím của hàm để xử lí theo yêu cầu đặt ra.

Ví dụ 1-19. Chương trình minh hoạ sử dụng *đối dòng lệnh*.

```
/* ****************************************************************************************** */
#include"stdio.h"
#include"conio.h"
/* Giả sử chương trình được biên dịch thành VD_18.exe trong thư mục C:\TC\ */
/* Trong chương trình ta có thể thay đổi cách xử lí của các tham số tuỳ ý */
/* ****************************************************************************************** */
void main(int i, char *str[])
{
    int k;
    printf("\nTen cua chuong trinh dang chay la: %s",str[0]);
    for(k=1;k<i;++k)
        printf("\nTham so thu  %d la:  %s", k, str[k]);
    getch();
    return;
}
/* ****************************************************************************************** */
```

Kết quả chạy chương trình như sau:

```
C:\TC\BIN\VD_18.exe /?  /is  /ie /u /m  /n
```

```
Ten cua chuong trinh dang chay la: C:\TC\BIN\VD_18.exe
Tham so thu 1 la: /?
Tham so thu 2 la: /is
Tham so thu 3 la: /ie
Tham so thu 4 la: /u
Tham so thu 5 la: /m
Tham so thu 6 la: /n
/* ************************************************************************************ */
```

1.4. MÔI TRƯỜNG LẬP TRÌNH TURBO C

1.4.1. Môi trường kết hợp

Có hai cách khác nhau để viết chương trình trong Turbo C. Một là hệ thống dòng lệnh C, theo đó việc *soạn thảo*, *biên dịch*, *liên kết* và *thực hiện* được gọi từ dòng lệnh DOS như những công việc riêng biệt do các chương trình riêng biệt đảm trách. Theo cách này thông thường các thao tác *biên dịch* và *liên kết* sẽ được thực hiện trong một tệp tin *.bat để giảm đỡ thời gian cho người lập trình. Hệ thống dòng lệnh C thường được sử dụng để liên kết các mô đun được viết bằng ngôn ngữ lập trình *Assembly* hoặc dùng để dịch các chương trình lớn. Trong giáo trình này không đề cập đến cách làm việc này.

Hệ thống thứ hai để tạo chương trình là dùng môi trường kết hợp của Turbo C (*Integrated Development Environment - IDE*). Trong môi trường này đã tích hợp tất cả các thao tác cần thiết để phát triển chương trình như *soạn thảo* tệp nguồn, *biên dịch* tệp nguồn thành các file đối tượng *.obj, liên kết các file đối tượng thành chương trình chạy được, sửa lỗi và chạy chương trình.

1. Cài đặt hệ thống

Cài đặt hệ thống lập trình Turbo C bằng cách chạy chương trình *install.exe* trong đĩa cài đặt thứ nhất. Tiến trình cài đặt sẽ hỏi tên ổ đĩa chứa các tệp nguồn (*mặc định là ổ A*) và đường dẫn tới thư mục sẽ chứa các tệp được cài đặt của Turbo C (*mặc định là C:\TC*). Tiến trình cài đặt sẽ sao chép các tệp tiêu đề **.h* vào thư mục con C:\TC\INCLUDE\, các tệp tin thư viện **.lib* và tệp đích **.obj* vào thư mục con C:\TC\LIB\ và các tệp tin khác của môi trường kết hợp Turbo C vào thư mục C:\TC bao gồm:

- TCC	Dùng để biên dịch theo dòng lệnh
- TLINK	Dùng để liên kết khi biên dịch theo dòng lệnh.
- MAKE	Dùng để quản lí tệp tin.
- GREP	Dùng để tìm chuỗi trong nhóm tệp tin.
- TOUCH	Dùng để cập nhật ngày tháng và giờ của tệp tin.
- CPP	Dùng cho các thao tác tiền xử lí.
- TCINST	Dùng để đặt cấu hình cho môi trường TC.
- TLIB	Dùng quản lí các chương trình thư viện.
- UNPACK	Dùng để bung tệp tin.
- OBJXREF	Dùng để tham khảo chéo các tệp tin đối tượng.
- BGIOBJ	Dùng để đổi trình điều khiển đồ hoạ sang tệp tin obj.

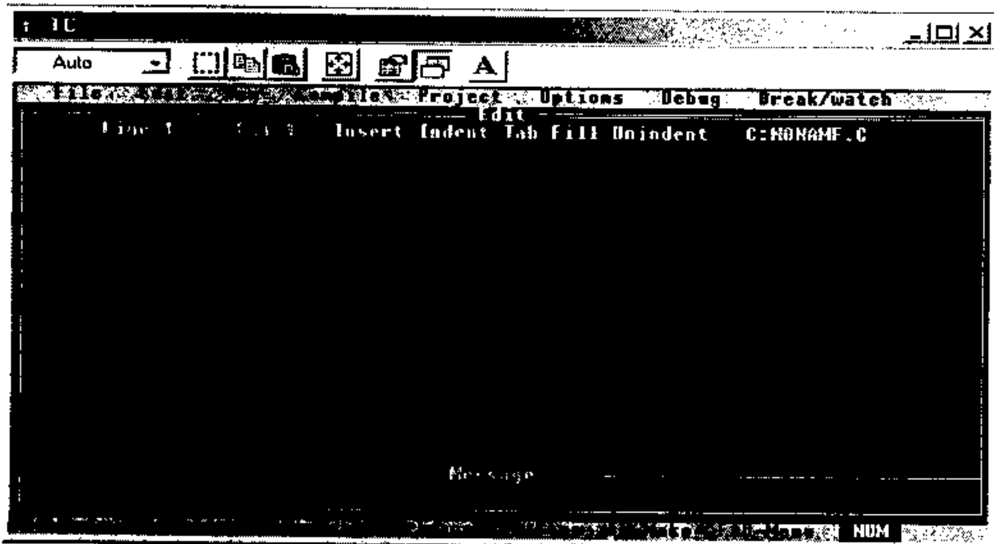
- THELP Dùng để trợ giúp trong quá trình sử dụng TC.
- README Dùng để đọc các tệp tin trợ giúp *readme*.

Trong đó, tệp tin quan trọng nhất là *TC.EXE*, nó chứa toàn bộ môi trường kết hợp của Turbo C. Để có thể chạy được *TC.EXE* từ bất kì thư mục nào dưới dòng lệnh *DOS* ta cần thêm dòng lệnh sau vào trong tệp *AUTOEXEC.BAT*:

SET PATH=C:\TC (nếu TC nằm trong thư mục khác thì phải chỉ rõ tên thư mục đó sau PATH=).

2. Sử dụng môi trường kết hợp của Turbo C

Để khởi động môi trường kết hợp của Turbo C, từ dấu nhắc của DOS ta gõ dòng lệnh sau đây: *TC.EXE* sau đó gõ ENTER. Kết quả ta thu được màn hình giao diện của Turbo C có dạng như sau:



Hình 1.1. Màn hình giao diện khi khởi động Turbo C.

Trước khi bắt đầu làm việc với môi trường kết hợp của Turbo C, ta cần xác lập lại cấu hình của môi trường trong tệp tin TCCONFIG.TC (nếu ta chưa thay đổi gì từ khi cài đặt thì có thể bỏ qua bước này). Để thực hiện được việc này ta chọn ***Option->Directories*** sau đó gõ đường dẫn của thư mục con \INCLUDE vào hộp thoại Include Directories (mặc định là C:\TC\INCLUDE), gõ đường dẫn của thư mục con \LIB vào hộp thoại Liblary Directories (mặc định là C:\TC\LIB). Bấm phím *ESC* rồi chọn *Save options* để lưu lại các thay đổi vào tệp cấu hình của môi trường.

Khi đã ở trong môi trường của Turbo C ta có thể thực hiện các thao tác sau:

- Bấm F1 để xem hướng dẫn về tình hình hiện tại.
- Bấm F10 để về Menu chính.
- Bấm đồng thời phím Alt và chữ cái đầu tiên của mỗi Menu để chọn từng Menu con. Ví dụ ta bấm Alt-F để vào Menu File, Alt-E để vào Menu Edit...

Các Menu chính trong môi trường Turbo C có thể được mô tả như sau:

Menu File có các chức năng sau:

. Load (F3): Nạp một tệp đã có vào để làm việc.

. Pick (Alt-F3): Lấy một tệp trong danh sách 8 tệp đã mở trước đó.

. New: Để tạo tệp mới với tên mặc định là Noname.c

. Save (F2): Ghi tệp đang soạn thảo ra đĩa.

. Write to: Ghi tệp đang soạn thảo ra đĩa với tên mới.

. Directory: Hiện nội dung thư mục hoặc tập hợp các tệp cần quan tâm.

. Change dir: Hiện nội dung thư mục hiện tại và cho phép đổi ổ đĩa hoặc thư mục.

. OS Shell: Tạm thời rời bỏ môi trường kết hợp để trở về câu lệnh của DOS. Muốn trở lại môi trường kết hợp ta gõ vào Exit.

. Quit (Alt-X): Thoát khỏi môi trường kết hợp của Turbo C để trở về *DOS*.

Menu Edit dùng để khởi động chương trình soạn thảo văn bản có sẵn của Turbo C. Màn hình soạn thảo thường luôn xuất hiện mặc định khi ta khởi động môi trường Turbo C.

Menu Run có các chức năng sau:

. Run (Ctrl-F9): Khởi động quá trình tự động biên dịch, liên kết và chạy chương trình thông qua các thông tin đã chuẩn bị sẵn trong chức năng Project name của Menu Project. Nếu không dùng Project name thì chương trình có tên trong hộp sáng Primary C file (Menu Compile) được xét. Nếu hộp sáng này rỗng thì chương trình đang soạn thảo hiện tại trong cửa sổ Editor được xét (*xem thêm phụ lục I để biết rõ hơn về Project name*).

. Program reset (Ctrl-F2): Lập lại chế độ thực hiện toàn bộ chương trình.

. Go to cursor (F4): Thực hiện chương trình cho đến dòng lệnh chứa con trỏ.

. Trace into (F7): Thực hiện từng lệnh một để gỡ rối.

. Step over (F8): Thực hiện từng câu lệnh và xem lời gọi hàm là một câu lệnh (*không chạy từng lệnh của hàm đó như F7*).

. User screen (Alt-F5): Trở lại màn hình sử dụng để xem kết quả do chương trình tạo ra. Bấm Enter sẽ trở về môi trường kết hợp của Turbo C.

Menu Compile có các chức năng sau:

. Compile to OBJ: Dịch một tệp *.*C* (theo thứ tự ưu tiên như đối với lệnh Run trong Menu Run) thành tệp *.Obj.

. Make EXE file: Dịch và liên kết để tạo thành tệp *.exe. Thứ tự chọn các tệp nguồn cũng tương tự như trên.

. Link EXE: Liên kết các tệp *.Obj và *.Lib để tạo thành tệp chương trình thực hiện *.exe.

. Build all: Dịch lại và liên kết để tạo thành tệp chương trình chạy được *.exe. Thứ tự các tệp nguồn cũng được xác định như trên.

. Primary C file: Dùng để chỉ định các tệp *.*C* cần được biên dịch và liên kết thành *.exe.

Menu Project có các chức năng sau:

. Project name: Định nghĩa một tệp project chứa thông tin về các tệp cần dịch và liên kết (*xem thêm phụ lục I*).

. Break make on: Xác định liệu việc dịch có bị dừng khi gặp Warning hoặc Error hay không.

. Auto dependencies: Rất ít được sử dụng.

. Clear Project: Dùng để xoá tệp Project và đặt lại cửa sổ Message.

. Remove message: Xoá bỏ cửa sổ Message.

Menu Option có các chức năng sau:

. Compile: Dùng để chọn mô hình bộ nhớ (*chọn Model*) và xác định độ dài cực đại của tên trong chương trình (*chọn Source*).

. Linker: Chọn cách thức liên kết.

. Environment: Thiết lập môi trường.

. Directories: Thiết lập các thư mục liên quan đến quá trình biên dịch.

. Argument: Đưa vào các tham số dòng lệnh cho chương trình chạy trong môi trường Turbo C (*nếu chương trình được viết dưới dạng đối dòng lệnh*).

. Save option: Ghi lại các thay đổi trong trong Menu Option.

. Retrieve option: Khôi phục lại thông tin option từ một tệp nào đó.

1.4.2. Các tệp tin thường được sử dụng trong ngôn ngữ lập trình C

- Các tệp tin tiêu đề *.h chứa trong thư mục \TC\INCLUDE\

- Các tệp tin thư viện *.lib chứa các đoạn chương trình đã được biên dịch mà nguyên mẫu của chúng được khai báo trong các tệp tiêu đề tương ứng. Khi cần sử dụng hàm nào thì chỉ đoạn chương trình của hàm đó được sử dụng (*chứ không phải toàn bộ tệp tin thư viện được chèn vào*). Trong ngôn ngữ lập trình C có 5 tệp tin thư viện chuẩn đó là *cs.lib*, *cc.lib*, *cm.lib*, *cl.lib* và *ch.lib* tương ứng với 5 mô hình bộ nhớ [14] của Turbo C là *small*, *compact*, *medium*, *large* và *huge*. Thông thường ta chỉ sử dụng mô hình *small*, do đó chỉ cần file *cs.lib* là đủ.

- Các tệp tin thư viện đích (*.obj): Đó là các tệp tin mà toàn bộ nội dung của nó sẽ được kết nối vào chương trình đang được biên dịch nếu chương trình đó cần đến các mã lệnh để thực hiện nhiều chức năng sau khi chạy như diễn dịch đối dòng lệnh... Trong ngôn ngữ lập trình C có 6 tệp tin thư viện loại này tương ứng với 6 mô hình bộ nhớ đó là *COT.obj, COS.obj, COC.obj, COM.obj, COL.obj và COH.obj* (*COT.obj tương ứng với mô hình Tiny*).

- Các tệp tin thư viện toán học: Dùng cho các phép toán dấu phẩy động. Có 5 tệp tin tương ứng với 5 mô hình bộ nhớ (mô hình Tiny dùng chung với small) đó là *maths.lib*, *mathc.lib*, *mathm.lib*, *mathl.lib* và *mathh.lib*. Ngoài ra, khi làm việc với các số dấu phẩy động ta cũng cần dùng thêm *fp87.lib* hay *emu.lib*. Tệp tin đầu tiên dùng cho máy có bộ tiền xử lí *Coprocessor 8087*, còn tệp tin thứ hai dùng để mô phỏng hoạt động của *8087* bằng phần mềm.

1.4.3. Cách viết và chạy chương trình trong môi trường kết hợp

Để tạo một tệp tin chương trình nguồn mới có tên VD.C ta có thể làm như sau, tại dấu nhắc của DOS ta gõ:

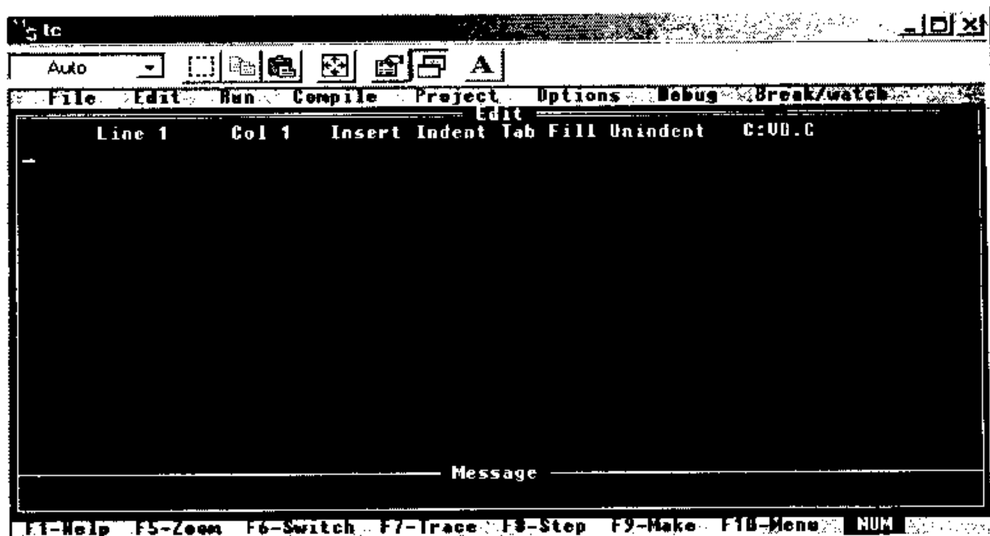
Tc.exe VD.C

hoặc khởi động môi trường kết hợp bằng câu lệnh Tc.exe rồi sau đó dùng Menu file->Write to để ghi thành tệp VD.C vào đĩa.

---

[14] Đó là cách tổ chức các đoạn chương trình trong bộ nhớ.

Theo cả hai cách ta đều nhận được màn hình *Edit* như sau:



Hình 1.2. Màn hình làm việc của Turbo C khi đang thao tác với tệp VD.C

Trong cửa sổ *Edit* của môi trường kết hợp ta có thể thực hiện các thao tác sau đây trong khi soạn thảo chương trình:

. Bấm F2 để lưu chương trình vừa soạn thảo.

. Dùng các phím mũi tên để di chuyển con trỏ.

. Bấm Ctrl + -> để chuyển con trỏ sang phải một từ.

. Bấm Ctrl + <- để chuyển con trỏ sang trái một từ.

. Bấm Home, End để chuyển con trỏ về đầu hoặc cuối dòng.

. Bấm Ctrl+Home để chuyển con trỏ về đầu trang màn hình.

. Bấm Ctrl+End để chuyển con trỏ về cuối trang màn hình.

. Bấm Page Up để chuyển con trỏ lên trang màn hình trước.

. Bấm Page Down để chuyển con trỏ xuống trang màn hình sau.

. Bấm Ctrl+Page Up để chuyển con trỏ về đầu chương trình.

. Bấm Ctrl+Page Down để chuyển con trỏ về cuối chương trình.

. Bấm BackSpace để xoá kí tự ngay trước con trỏ.

. Bấm Delete để xoá kí tự ngay sau con trỏ.

. Bấm Ctrl +T để xoá từ đang chứa con trỏ.

. Bấm Ctrl +Q+Y để xoá từ vị trí con trỏ đến cuối dòng.

. Bấm Ctrl +Y để xoá dòng đang chứa con trỏ.

. Bấm Ctrl +N để chèn thêm dòng trống vào trước dòng chứa con trỏ.

. Bấm Insert để chuyển đổi chế độ chèn và đè.

. Bấm Ctrl+K+B để đánh dấu đầu khối.

. Bấm Ctrl+K+K để đánh dấu cuối khối.

. Bấm Ctrl+K+C để copy khối đã được đánh dấu.

. Bấm Ctrl+K+V để dán khối đã được copy.

. Bấm Ctrl+K+Y để xoá khối đã được đánh dấu.

. Bấm Ctrl+K+P để in khối ra máy in.

. Bấm Ctrl+K+W để ghi khối đã được đánh dấu ra đĩa.

. Bấm Ctrl+K+R để chèn nội dung file trên đĩa vào vị trí con trỏ.

. Bấm Ctrl+K+H để bỏ việc đánh dấu khối.

. Bấm Ctrl+Q+F để tìm kiếm văn bản trong nội dung file đang soạn thảo.

. Bấm Ctrl+Q+A để tìm kiếm và thay thế văn bản trong nội dung file đang soạn thảo.

. Bấm Ctrl+O+I để chuyển đổi chế độ căn lề trái.

. Bấm Ctrl+O+U để chuyển đổi chế độ căn lề trái khi sử dụng phím xoá BackSpace.

Chương trình sau khi viết xong có thể biên dịch, liên kết và chạy tự động bằng cách bấm cùng lúc *Ctrl+F9*. Nếu chương trình viết đúng nó sẽ được biên dịch, liên kết và chạy ngay trong môi trường kết hợp của Turbo C. Để có thể dừng màn hình để xem kết quả nhận được thì ở cuối hàm *main*, trước câu lệnh *return* ta cho thêm câu lệnh *getch()*. Hoặc ta có thể dùng chức năng *User screen* bằng cách bấm *Alt-F5* để xem kết quả khi chương trình thực hiện xong.

1.4.4. Sửa lỗi cú pháp và gỡ rối chương trình

Các chức năng sửa lỗi cú pháp và gỡ rối chương trình là một phần trong môi trường kết hợp của Turbo C. Khi biên dịch một chương trình, nếu gặp lỗi, trình biên dịch sẽ thông báo các lỗi mắc phải trong cửa sổ *Message* ở cuối màn hình (muốn *cho cửa sổ này hiện toàn bộ màn hình ta bấm F5*). Khi chuyển con trỏ tới một lỗi nào đó trong danh sách lỗi nhận được thì dòng lệnh tại nơi phát sinh ra lỗi đó cũng được đánh dấu, ta có thể bấm *Enter* hoặc *Alt-E* để chuyển con trỏ tới chỗ có lỗi và sửa theo thông báo gợi ý của trình biên dịch. Các lỗi này đều là lỗi về mặt cú pháp. Có những trường hợp, khi biên dịch không có lỗi nhưng chương trình bị lặp vô hạn hoặc cho kết quả sai. Các lỗi loại này gọi là các lỗi về mặt *logic*. Để có thể sửa chữa các lỗi *logic* trong Turbo C ta cần sử dụng chức năng gỡ rối chương trình của môi trường kết hợp đó là chức năng chạy chương trình từng bước (*F7 hoặc F8*) và chức năng kiểm tra các giá trị của biến (*cửa sổ Watch*). Muốn làm xuất hiện cửa sổ *Watch* bên dưới màn hình ta vào *Menu* con *Windows* rồi sau đó vào *Watch*. Để chuyển đổi giữa các cửa sổ ta dùng *F6*. Trong quá trình gỡ rối, để quan sát sự biến đổi giá trị của một biến nào đó ta phải đưa tên của biến đó vào trong cửa sổ Watch bằng cách chọn "*Add Watch*" trong Menu *Break/watch* hoặc bấm *Ctrl-F7* sau đó nhập tên biến và bấm *Enter*. Mỗi lần nhập như vậy chỉ có thể cho một biến mà thôi, muốn nhập cho nhiều biến ta phải lặp lại quá trình trên.

Để có thể tìm được nguyên nhân của các lỗi *logic*, thông thường ta cho chương trình chạy từng bước bằng F7 hoặc F8 sau đó quan sát sự thay đổi giá trị của các biến tương ứng trong cửa sổ *Watch*, từ đó có thể tìm được nguyên nhân sai hỏng.

Trong quá trình gỡ rối ta cũng có khả năng thay đổi giá trị của một biến nào đó để quan sát những thay đổi trên các biến khác bằng cách dùng cửa sổ

EVALUATION trong *Menu Debug* hoặc bấm *Ctrl-F4* khi đó trên màn hình sẽ hiện ra cửa sổ như hình 1.3.

```
┌──────────────────── Evaluate ────────────────────┐
│ _                                                │
└──────────────────────────────────────────────────┘
┌──────────────────── Result ──────────────────────┐
│                                                  │
└──────────────────────────────────────────────────┘
┌──────────────────── New value ───────────────────┐
│                                                  │
└──────────────────────────────────────────────────┘
```

Hình 1.3. Cửa sổ Evaluation dùng để thiết lập lại giá trị cho các biến.

Nhập tên biến cần thay đổi giá trị trong hộp Evaluate, khi đó giá trị hiện thời của biến sẽ hiện trong hộp *Result*. Nhập giá trị mới của biến vào hộp *New value* sau đó trở về cửa sổ *Watch* để quan sát sự thay đổi giá trị của các biến khác bằng bấm *ESC*, hoặc có thể xem trực tiếp trong cửa sổ *Evaluation* bằng cách gõ tên các biến mới vào hộp *Evaluate* rồi quan sát giá trị trong hộp *Result*.

Để kết thúc quá trình gỡ rối, bấm *Ctrl-F2* hoặc vào Menu Run chọn *Program reset*. Khi đó bộ nhớ dùng cho việc gỡ rối sẽ được giải toả.

Chú ý:

Chương trình biên dịch có thể sinh ra những thông báo lỗi không chính xác từ một lỗi trước đó. Do đó sau khi sửa xong một số lỗi nào đó ta nên biên dịch lại (*bấm F9*) để bỏ đi những dòng thông báo thừa trong cửa sổ *Message*.

1.5. MỘT SỐ VÍ DỤ MINH HOẠ VÀO RA

Ví dụ 1-20. Lập trình để máy nói chuyện với người sử dụng. Máy "nói" bằng các câu lệnh trên màn hình, còn người sử dụng "nói" bằng cách gõ vào từ bàn phím. Nội dung cuộc nói chuyện tuỳ thích.

```
/* ******************************************************************************** */
#include "stdio.h"
#include "conio.h"
/* ******************************************************************************** */
int main()
{
    char traloi[20], ten[25];
    printf("\nHom nay troi dep qua. Ban tu dau den vay ?\n");
    gets(traloi);
    printf("Toi cung co anh ban o  %s", traloi);
    printf("\nBan ten la gi ?\n");
    gets(ten);
    printf("Chao ban  %s", ten);
    printf("\nBan khoe chu ?\n");
    gets(traloi);
    printf("Toi dang ban di hoc, hen ban %s lan sau nhe.", ten);
    printf("\nTam biet!");
    getch();
    return 0;
```

```
}
/* ********************************************************************************** */
```

Kết quả chạy chương trình:

```
Hom nay troi dep qua. Ban tu dau den vay?
Hai Phong
Toi cung co anh ban o Hai Phong
Ten ban la gi?
Nguyen Van A
Chao ban Nguyen Van A
Ban khoe chu ?
Toi dang ban di hoc, hen ban Nguyen Van A lan sau nhe.
Tam biet!
/* ********************************************************************************** */
```

Ví dụ 1-21. /*Chương trình minh hoạ nếu không dùng đúng kí tự chuyển dạng trong *printf()* thì kết quả in ra sẽ sai*/

```
/* ********************************************************************************** */
#include "stdio.h"
#include "conio.h"
/* ********************************************************************************** */
int main()
{
    int x=10;
    long y=3478925;
    printf("\nx=%10ld\ny=%10d", x, y);
    getch();
    return 0;
}
/* ********************************************************************************** */
```

Kết quả chạy chương trình:

```
x=361562122
y=         53
/* ********************************************************************************** */
```

Ví dụ 1-22. /*Chương trình dùng *sai đặc tả* trong câu lệnh *scanf()*, mặc dầu không bị báo lỗi nhưng khi chạy cho kết quả sai*/

```
/* ********************************************************************************** */
#include "stdio.h"
#include "conio.h"
/* ********************************************************************************** */
int main()
{
    int a;
    long b;
    float x;
    double y;
    printf("\nNhap a,  b,  x,  y\n");
    scanf("%ld%d%lf%f", &a, &b, &x, &y);
```

```
    printf("\na=%10d\nb=%10ld\nx=%10.2f\ny=%10.2f", a, b, x, y);
    getch();
    return 0;
}
/* ********************************************************************************************* */
```

Kết quả chạy chương trình:

```
    Nhap a, b, x, y
    2346 447895 347 246
    a= 2346
    b= 119592343
    x= 0.00
    y= 5.549683991018940230000000000000000000000e+191
/* ********************************************************************************************* */
```

1.6. CÂU HỎI VÀ BÀI TẬP

1. Điểm khác biệt giữa tên trong ngôn ngữ lập trình C và trong Pascal là gì? cho ví dụ.
2. Đặt 10 tên hợp lệ trong ngôn ngữ lập trình C.
3. Hãy viết 15 tên khác nhau từ các chữ cái A và a.
4. Ngôn ngữ lập trình C có những đặc trưng gì?
5. Trong ngôn ngữ lập trình C có những kiểu dữ liệu chuẩn nào? cách sử dụng của từng loại đó?
6. Khối lệnh dùng để làm gì, vai trò của nó trong chương trình ?
7. Trình bày về phạm vi sử dụng và thời gian tồn tại của một biến. Vai trò của chúng trong chương trình.
8. Có những loại hằng nào? cách định nghĩa và sử dụng của tên hằng và biến hằng.
9. Đặc trưng của một câu lệnh trong ngôn ngữ lập trình C là gì ?
10. Hãy thực hiện các ví dụ mẫu bằng máy tính.
11. Tìm chỗ sai trong đoạn chương trình sau đây:

    ```
    #include "stdio.h"
    main()
    {
        printf("\na=%10.0f, b=%10d, c=%10ld, d=%10d", -3456, 25e3,
        4635, 456398461);
    }
    ```

    Sửa lại cho đúng rồi thực hiện trên máy tính.

12. Viết chương trình in một dòng với nội dung sau:

    N=365

Bằng cách sử dụng các loại hằng khác nhau: hằng dấu phẩy động, hằng nguyên, hằng long, hằng nguyên hệ 8, hằng nguyên hệ 16, hằng kí tự và hằng xâu kí tự.

13. Tìm các chỗ sai trong đoạn chương trình sau:

```
{
        Float a, b=2;
        {
                int a=10;
                float c=a*b, d=a+b;
        }
        printf("\na=%10.2d, b=%10.2f, c=%f, d=%10.2f", a, b, c, d);
}
```

Sửa chữa và bổ sung để được một chương trình hoàn thiện, sau đó thực hiện trên máy tính.

14. Cho 3 số A=555; B=-4.3; C=2100000000. Hãy viết chương trình đưa các dữ liệu này vào máy bằng hằng sau đó đưa ra màn hình theo quy cách dùng 14 vị trí trong đó có 3 số lẻ nếu là số thực.
15. Viết chương trình nhập họ tên, ngày tháng năm sinh của một học sinh vào máy rồi đưa ra màn hình, mỗi dữ liệu trên một dòng.
16. Viết chương trình nhập một kí tự chữ thường từ bàn phím, sau đó đưa ra màn hình ở dạng chữ hoa.
17. Viết lại chương trình ví dụ 1-20 với nội dung phong phú hơn.
18. Thực hành sử dụng các tiện ích gỡ rối của Turbo C để tìm ra chỗ sai của chương trình dùng để hoán vị nội dung của hai biến sau đây:

```
#include "stdio.h"
#include "conio.h"
void HoanVi(int, int);
int main(void)
{
    int a, b;
    printf("\nHay nhap hai so nguyen a va b \n");
    scanf("%d%d",&a, &b);
    printf("\nGia tri cua a truoc khi hoan vi la:\na= %10d", a);
    printf("\nGia tri cua b truoc khi hoan vi la:\nb= %10d", b);
    HoanVi(a,b);
    printf("\nGia tri cua a sau khi hoan vi la:\na= %10d", a);
    printf("\nGia tri cua b sau khi hoan vi la:\nb= %10d", b);
    getch();
    return 0;
}
void HoanVi(int a, int b)
{
    int tg;
    tg= a;
    a=b;
    b=tg;
}
```

Chương 2
BIỂU THỨC

MỤC TIÊU CỦA CHƯƠNG NÀY

- *Hiểu khái niệm về biểu thức trong ngôn ngữ lập trình C.*
- *Hiểu và sử dụng thành thạo các phép toán trong ngôn ngữ C, có thể biến đổi một biểu thức toán học thường gặp thành biểu thức đúng mà ngôn ngữ C có thể hiểu được.*
- *Hiểu và sử dụng thành thạo các cấu trúc điều khiển (toán tử điều khiển) của ngôn ngữ C.*

2.1. KHÁI NIỆM

Biểu thức là một khái niệm rất quan trọng và không thể thiếu được của bất kì một ngôn ngữ lập trình nào dùng để diễn đạt một công thức, một quy trình tính toán nào đó. Trong ngôn ngữ lập trình C, ***biểu thức*** được hiểu là sự kết hợp giữa các ***toán tử*** và các ***toán hạng*** theo một trật tự và một số quy tắc nhất định nhằm diễn đạt một công thức toán học nào đấy. ***Toán hạng*** là các đại lượng mà có một giá trị xác định nào đó. Trong ngôn ngữ lập trình C, ***toán hạng*** có thể là các hằng, biến, phần tử mảng và các lời gọi hàm mà có trả về một giá trị cụ thể. Còn ***toán tử*** là các thao tác xử lí trên các ***toán hạng*** theo các quy tắc nhất định để đạt được mục đích mong muốn. Ngôn ngữ lập trình C có hai loại ***toán tử***, đó là các ***phép toán*** và các ***toán tử điều khiển***. Các ***phép toán*** chủ yếu dùng để xây dựng lên các ***biểu thức số học***, ***biểu thức quan hệ*** và ***logic***. Còn các ***toán tử điều khiển*** dùng để xây dựng lên các kiểu cấu trúc cơ bản của một chương trình máy tính (*cấu trúc rẽ nhánh và cấu trúc lặp*).

2.2. CÁC TOÁN TỬ TRONG NGÔN NGỮ LẬP TRÌNH C

2.2.1. Các Phép toán (*các phép xử lí*)

Trong ngôn ngữ lập trình C, các phép toán rất đa dạng và phong phú từ các phép toán số học, quan hệ và logic xử lí trên các kiểu dữ liệu đã có cho đến các phép thao tác ***bit*** chỉ xử lí dưới dạng các bit nhị phân. Đặc biệt C còn đưa thêm vào ***toán tử điều kiện*** và các chức năng mới cho ***toán tử gán*** tạo cho C có thêm những tiềm năng mới trong việc giải quyết các bài toán thực tế.

1. Các phép toán số học

Các phép toán số học là các phép toán dùng để viết các công thức toán học, bao gồm những phép toán sau đây:

Bảng 2.1. Các phép toán số học trong ngôn ngữ lập trình C

Tên phép toán	**Biểu diễn**	**Chức năng**
Phép cộng	BT1+BT2 [15]	Phép toán hai ngôi dùng để thực hiện cộng giá trị của BT1 với BT2.
Phép trừ	BT1-BT2	Phép toán hai ngôi dùng để thực hiện phép trừ giá trị của BT1 cho BT2.
Phép nhân	BT1*BT2	Phép toán hai ngôi dùng để thực hiện nhân giá trị của BT1 với BT2.
Phép chia	BT1/BT2	Phép toán hai ngôi dùng để thực hiện phép chia giá trị của BT1 cho BT2. Với điều kiện BT2 khác không.
Phép lấy phần dư	BT1%BT2	Phép toán hai ngôi dùng để thực hiện lấy phần dư của phép chia BT1 cho BT2. Trong đó BT1 và BT2 là các biểu thức nguyên.
Phép đổi dấu	- BT	Phép toán một ngôi dùng để đổi dấu giá trị của BT.

Chú ý:

- Nếu phép chia thực hiện trên hai số nguyên mà số bị chia không chia hết cho số chia thì kết quả nhận được sẽ chỉ là phần nguyên của phép chia đó. Ví dụ:

10/3 sẽ có giá trị là 3.
9/3 sẽ có giá trị là 3.

Để có thể nhận được kết quả đúng của phép 10/3 ta cần có sự chuyển kiểu sang số thực (*xem mục 2.2.1. phần 8*) như sau:

(float)10/3 sẽ cho kết quả đúng là 3.3333333...

- Phép lấy phần dư chỉ có thể thực hiện trên số nguyên mà thôi.

- Các phép +, - có cùng thứ tự ưu tiên nhưng nhỏ hơn thứ tự ưu tiên của *, / và %. Ba phép toán này lại có thứ tự nhỏ hơn phép '-' (*một ngôi*). Các phép toán số học sẽ được thực hiện từ trái qua phải và các phép có thứ tự ưu tiên cao hơn được thực hiện trước, các phép có giá trị ưu tiên thấp hơn sẽ được thực hiện sau.

2. Các phép toán quan hệ

Các phép toán quan hệ là các phép toán dùng để so sánh giá trị của các biểu thức với nhau phục vụ cho việc ra các quyết định trong chương trình. Khác với ***Pascal,*** các phép toán quan hệ trong ngôn ngữ lập trình C trả về giá trị *1* (*tương ứng với true*) nếu phép so sánh là đúng, ngược lại trả về giá trị *0* (*tương ứng với false*).

[15] BT1, BT2, BT có thể là các biểu thức số học hoặc biểu thức quan hệ và logic.

Bảng 2.2. Các phép toán quan hệ thông dụng trong ngôn ngữ lập trình C

Tên phép toán	Biểu diễn	Chức năng	Ví dụ
Phép so sánh lớn hơn	BT1 > BT2 [16]	So sánh giá trị của BT1 với BT2, nếu BT1 lớn hơn BT2 biểu thức sẽ trả về giá trị 1, ngược lại trả về giá trị 0.	3>6 có giá trị 0. (10+2*3)>3 có giá trị 1.
Phép so sánh nhỏ hơn	BT1 < BT2	So sánh giá trị của BT1 với BT2, nếu BT1 nhỏ hơn BT2 biểu thức sẽ trả về giá trị 1, ngược lại trả về giá trị 0.	3<6 có giá trị 1. 20+2<2 có giá trị 0.
Phép so sánh bằng	BT1 == BT2	So sánh giá trị của BT1 với BT2, nếu BT1 bằng BT2 biểu thức sẽ trả về giá trị 1, ngược lại trả về giá trị 0.	2+3==1+4 cho giá trị 1.
Phép so sánh khác (không bằng)	BT1 != BT2	So sánh giá trị của BT1 với BT2, nếu BT1 khác BT2 biểu thức sẽ trả về giá trị 1, ngược lại trả về giá trị 0.	3*(3+2) != 5 cho giá trị 1.
Phép so sánh lớn hơn hoặc bằng	BT1 >= BT2	So sánh giá trị của BT1 với BT2, nếu BT1 lớn hơn hay bằng BT2 biểu thức sẽ trả về giá trị 1, ngược lại trả về giá trị 0.	10+3*5>=7 cho giá trị 1. 20>=20 cho giá trị 1.
Phép so sánh nhỏ hơn hoặc bằng	BT1 <= BT2	So sánh giá trị của BT1 với BT2, nếu BT1 nhỏ hơn hay bằng BT2 biểu thức sẽ trả về giá trị 1, ngược lại trả về giá trị 0.	(a+b)*(c+d)<=12 với a, b, c và d lần lượt là 1, 3, 5 và 7 sẽ có giá trị 0.

Chú ý:

- Bốn phép đầu có cùng thứ tự ưu tiên, hai phép sau cũng có cùng thứ tự ưu tiên nhưng thấp hơn bốn phép đầu tiên.
- Các phép toán quan hệ có thứ tự ưu tiên nhỏ hơn các phép toán số học.
- Một biểu thức quan hệ chỉ có thể có giá trị 1 (*đúng*) hoặc 0 (*sai*).

Ví dụ 2-1. Xét biểu thức quan hệ *e == a+b>=c+d* thì phép *a+b* được thực hiện trước tiên sau đó đến phép *c+d*, tiếp theo kết quả của hai phép cộng sẽ được so sánh với nhau thông qua phép so sánh '>='. Cuối cùng giá trị của ***e*** mới được so sánh với kết quả vừa nhận được trong phép so sánh '>=' thông qua phép so sánh '=='. Kết quả này chính là kết quả cuối cùng của biểu thức quan hệ vừa xét.

3. Các phép toán logic

Các phép toán logic là các phép toán thường dùng kết hợp với các phép quan hệ để xây dựng lên các ***biểu thức quan hệ và logic***. Biểu thức quan hệ và logic là biểu thức chỉ có giá trị 1 (*đúng*) hoặc 0 (*sai*) như *ví dụ 2-1*. Trong ngôn ngữ lập trình C có ba phép toán logic thông dụng là phép phủ *định một ngôi*, *phép hội* và *phép hoặc*. Chức năng và cách biểu diễn của chúng được mô tả trong bảng 2.3.

[16] BT1 và BT2 cũng có thể là các biểu thức số học hoặc biểu thức quan hệ và logic.

Bảng 2.3. Các phép toán logic thông dụng trong ngôn ngữ lập trình C

Tên phép toán	Biểu diễn	Chức năng	Ví dụ
Phép phủ định một ngôi	! BT	Nếu BT có giá trị khác 0 sẽ cho giá trị 0. Nếu BT có giá trị bằng 0 sẽ cho giá trị1.	! (2+5) cho giá trị 0.
Phép hội (phép AND)	BT1 && BT2	Nếu BT1 bằng 0 và BT2 bằng 0 sẽ cho giá trị 0 Nếu BT1 bằng 0 và BT2 *khác* 0 sẽ cho giá trị 0 Nếu BT1 *khác* 0 và BT2 bằng 0 sẽ cho giá trị 0 Nếu BT1 *khác* 0 và BT2 *khác* 0 sẽ cho giá trị 1	0 && (2+3) cho giá trị 0 (2+6) && 8 cho giá trị 1
Phép hoặc (phép OR)	BT1 || BT2	Nếu BT1 bằng 0 và BT2 bằng 0 sẽ cho giá trị 0 Nếu BT1 bằng 0 và BT2 *khác* 0 sẽ cho giá trị 1 Nếu BT1 *khác* 0 và BT2 bằng 0 sẽ cho giá trị 1 Nếu BT1 *khác* 0 và BT2 *khác* 0 sẽ cho giá trị 1	0 || (5-5) cho giá trị 0 0 || (10+3) cho giá trị 1

Chú ý:

Phép phủ định một ngôi '!' có mức ưu tiên cao hơn các phép quan hệ và số học, nhưng các *phép hội* và *phép hoặc* lại có mức ưu tiên thấp hơn các phép toán quan hệ (*xem thêm mục 10 để biết thêm về thứ tự ưu tiên của các phép toán*).

4. Các phép thao tác bit

Đây là các thao tác thường hay dùng trong hợp ngữ (*ngôn ngữ Assembly*). Trong ngôn ngữ lập trình C các phép thao tác bit chỉ tiến hành xử lí trên từng ***bit*** nhị phân của một số nguyên (*không dùng cho số thực*) và đôi khi chúng được sử dụng tương đối hữu dụng. Các phép thao tác Bit bao gồm:

Phép toán	Ý nghĩa	Ví dụ
&	Phép AND theo Bit	x&y
|	Phép OR theo Bit	x|y
^	Phép XOR	x^y
<<	Phép dịch trái	a<<4
>>	Phép dịch phải	a>>4
~	Phép đảo Bit	~x [17]

Bảng 2.4. Hoạt động của các phép thao tác bit trong ngôn ngữ lập trình C

1&1 bằng 1	1|1 bằng 1	1^1 bằng 0	!1 bằng 0
1&0 bằng 0	1|0 bằng 1	1^0 bằng 1	!0 bằng 1
0&1 bằng 0	0|1 bằng 1	0^1 bằng 1	a<<n bằng $a*2^n$
0&0 bằng 0	0|0 bằng 0	0^0 bằng 0	a>>n bằng $a/2^n$

Chú ý:

Cần phân biệt các phép thao tác Bit '&', '|', '!' với các phép '&&', '||' và '-'. Ví dụ với biến nguyên a=0x0fff (giá trị 4095 trong hệ thập phân) và biến nguyên b=0x00ff (có giá trị 255 trong hệ thập phân) thì:

a&&b cho kết quả 1   a&b cho kết quả 0x00ff

[17] *x,y* là các Bit nhị phân của một biến nguyên, *a* là một biến nguyên. *BT*, *BT1*, *BT2* cũng có thể là các biểu thức số học hoặc biểu thức quan hệ và logic.

```
a||b cho kết quả 1          a|b cho kết quả 0x0fff
!a cho kết quả 0xf000       -a cho kết quả 0xf001(-4095)
!b cho kết quả 0xff00       -b cho kết quả 0xff01 (-255)
```

Ví dụ 2-2. Để trích byte thấp và byte cao của một biến nguyên ***a*** ta có thể làm như sau:

```
LowByte=a&0x00ff; /*nửa cao của a sẽ chỉ có giá trị 0, nửa thấp sẽ giữ
                    nguyên giá trị của nó*/
HighByte=a>>8;
printf("\nGia tri cua Byte cao la: %x",HighByte);
printf("\nGia tri cua Byte thap la: %x",LowByte);
```

Ví dụ 2-3. Để kiểm tra Bit thứ 6 của một số nguyên ***a*** bằng hay khác 0 ta có thể tiến hành theo hai cách như sau:

- ***Cách 1***:

```
b=a>>6;/*Đưa Bit thứ 6 của biến a về vị trí Bit 1*/
if(0x01 &b)
      printf("\nKhac khong");
else
      printf("\nBang khong");
```

- ***Cách 2***: Thiết lập ***mặt nạ Bit*** như sau:

```
if(0x0040&a)/*Nếu bit 6 khác 0*/
      printf("\nKhac khong");
else
      printf("\nBang khong");
```

5. Các phép tăng / giảm

Các phép tăng (giảm) là các phép toán một ngôi trong ngôn ngữ lập trình C dùng để tăng (giảm) các biến (nguyên hoặc thực) lên (xuống) 1 giá trị. Bao gồm các phép sau đây:

Bảng 2.5. Các phép tăng / giảm trong ngôn ngữ lập trình C

Tên phép toán	Biểu diễn	Chức năng
Phép tăng trước	++i [18]	Sau khi thực hiện câu lệnh giá trị của i tăng lên 1
Phép tăng sau	i++	Sau khi thực hiện câu lệnh giá trị của i tăng lên 1
Phép giảm trước	--i	Sau khi thực hiện câu lệnh giá trị của i giảm đi 1
Phép giảm sau	i--	Sau khi thực hiện câu lệnh giá trị của i giảm đi 1

Chú ý:

- Cần phân biệt sự khác nhau giữa việc tăng trước và tăng sau (*cũng như giảm trước và giảm sau*). Trong phép tăng sau (*giảm sau*) thì biến ***i*** tăng (*giảm*) sau khi giá trị của nó đã được sử dụng. Ngược lại, trong phép tăng trước (*giảm trước*) thì biến ***i*** tăng (*giảm*) trước khi giá trị của nó đã được sử dụng. Còn kết quả thực hiện của các phép toán trên thì hoàn toàn giống nhau.

Ví dụ 2-4. Xét hai đoạn chương trình sau:

```
int i=10, x;                int i=10, x;
```

[18] i là một biến nguyên hoặc thực.

x=i++;	x=++i;
x sẽ có giá trị là 10; còn i sẽ có giá trị là 11	x sẽ có giá trị là 11; còn i sẽ có giá trị là 11

- Không nên sử dụng các phép tăng giảm quá tuỳ tiện trong các biểu thức vì có thể sẽ dẫn đến các kết quả sai không mong muốn.

6. Phép lấy địa chỉ và kích thước biến

Chúng ta đã biết rằng, mỗi biến sau khi khai báo sẽ được cấp phát một vùng nhớ gồm một số *Byte* liên tiếp (*số lượng các Byte phụ thuộc vào loại biến*). Địa chỉ của *Byte* nhớ đầu tiên trong bộ nhớ sẽ được coi là địa chỉ của biến đó. Để truy nhập đến vùng nhớ của biến ta có thể truy nhập thông qua tên biến, nhưng đôi khi bắt buộc ta phải truy nhập thông qua địa chỉ của biến (*ví dụ như trong câu lệnh scanf*). Muốn biết được địa chỉ của một biến *x* nào đó trong bộ nhớ ta có thể viết &x.

Khi muốn biết kích thước (*tính theo Byte*) của một biến hay một kiểu dữ liệu ta có thể dùng toán tử *sizeof* theo cú pháp như sau:

```
sizeof(Kiểu_dữ_liệu); (19)
hoặc sizeof(Tên_biến);
```

Ví dụ 2-5. Cách dùng toán tử *sizeof*. Để xác định số phần tử ***n*** của một mảng A trong cơ chế khởi đầu ta làm như sau:

```
double A[]={10., 69.3,39.4};
int n=sizeof(A)/sizeof(double);
```

7. Phép gán và biểu thức

Phép gán là phép dùng để đưa giá trị của một biểu thức nào đó vào trong một biến tương thích theo cú pháp như sau:

```
Biến=BT;
```

Trước tiên, máy sẽ tiến hành tính giá trị của biểu thức BT, sau đó mới gán giá trị tính được cho biến ***Biến*** (do mức ưu tiên của toán tử gán gần như thấp nhất, chỉ cao hơn toán tử ','. *Xem thêm 2.2.1. mục 10*).

Ví dụ 2-6. Xét đoạn chương trình sau:

```
float a, b=3.5;
a=b*(b-1.5)+10;
```

Kết quả tính toán của biểu thức *b*(b-1.5)+10* sẽ được gán cho biến *a* (*17*).

Xét đoạn chương trình sau:

```
float x, y, z, a, b;
a=b=1999;
z=(y=3)*(x=10);
```

[19] Kiểu dữ liệu có thể là kiểu chuẩn như int, float, … hoặc là các kiểu tự định nghĩa.

Kết quả là biến *a* và biến *b* được gán cùng một giá trị là *1999*, còn biến *x* được gán giá trị *10*, biến *y* được gán giá trị *3* và kết quả của phép tính *3*10* sẽ được gán cho biến *z*.

Đối với một số phép gán đặc biệt mà biến được gán có mặt cả trong hai vế của biểu thức gán, ví dụ như:

```
int i,j,k;
i=i+2;          có thể viết ngắn hơn như sau  ──────►   i+=2;
j=j-1;                                                  j-=1;
k=k*(i+j);                                              k*=i+j;
```

Chú ý:

- Cần phân biệt phép gán '=' với dấu đẳng thức trong toán học.

- Kiểu của biểu thức *BT* ở trên phải ***tương thích*** với kiểu của biến được gán giá trị. Nếu sự tương thích này không thoả mãn thì chương trình sẽ phát sinh một lỗi. Sự tương thích ở đây có nghĩa là kiểu của biểu thức *BT* có thể được chuyển đổi một cách tự động sang kiểu của biến gán theo nguyên tắc kiểu *int* có thể chuyển thành kiểu *float*, kiểu *float* có thể chuyển thành kiểu *int* bằng cách chặt cụt phần thập *phân*, kiểu *double* có thể chuyển thành kiểu *float* bằng cách làm tròn, kiểu *long* có thể chuyển thành kiểu *int* bằng cách cắt bỏ một vài chữ số…

- Đối với dạng viết ngắn có thể áp dụng cho các *phép toán số học* và các *phép thao tác Bit hai ngôi* như +, -, *, / , %, <<, >>, &, ^ và |. Ví dụ để gán số dư của phép chia *a* cho *b* ta có thể viết như sau: a%=b; khi đó *a* sẽ chứa số dư cần tìm. Hoặc để xóa biến nguyên *e* về không ta có thể viết e^=e;

8. Chuyển đổi kiểu giá trị trong các biểu thức

Việc chuyển kiểu thường xảy ra trong các trường hợp: hoặc biểu thức gồm các toán hạng khác kiểu, hoặc gán các đối tượng khác kiểu cho nhau, hoặc truyền tham số khác kiểu cho các hàm và trả về giá trị khác kiểu bằng câu lệnh *return*. Việc chuyển kiểu này ***nếu*** có thể tiến hành tự động thì sẽ tuân theo sơ đồ sau:

Char->int->long->float->double->long double

Có nghĩa là nếu hai toán hạng trong một phép toán có kiểu khác nhau thì kiểu thấp hơn (*có phạm vi nhỏ hơn*) sẽ được nâng thành kiểu cao hơn cho phù hợp với toán hạng kia trước khi phép toán được tiến hành xử lí.

Ví dụ 2-7. Khi thực hiện các phép tính sau:

1.5*(11/3) sẽ cho kết quả là 4.5 (kiểu float)
1.5*11/3 sẽ cho kết quả là 5.5 (kiểu float)

Trong những trường hợp ta muốn điều khiển quá trình chuyển đổi theo ý mình thì có thể dùng một phép thao tác đặc biệt đó là phép ***ép kiểu***.

Phép ép kiểu là phép dùng để chuyển đổi giá trị của một biểu thức từ *kiểu* này sang *kiểu* khác theo chỉ định cụ thể. Cú pháp của phép ép kiểu như sau:

(Kiểu) BT;

Trong đó ***Kiểu*** là kiểu cần chuyển đến, *BT* là biểu thức cần được chuyển thành ***Kiểu*** theo những nguyên tắc như đối với việc chuyển đổi tự động. Ví dụ câu lệnh (int)(10.3) sẽ cho kết quả là 10.

Chú ý:

- Phép ép kiểu không làm thay đổi kiểu và giá trị của chính bản thân biến bị ép kiểu. Ví dụ nếu ***a*** là một biến kiểu *float* thì sau phép (int) a thì ***a*** vẫn là biến kiểu ***float*** nhưng (int) a lại có giá trị kiểu số nguyên.

- Để chuyển đổi giá trị thực sang giá trị nguyên, nên dùng (int)(BT+0.5).

- Trong quá trình chuyển đổi cần chú ý đến thứ tự ưu tiên của các phép toán. Ví dụ xét hai biểu thức sau:

(int)1.6*100 sẽ cho kết quả là 100

(int)(1.6*100) sẽ cho kết quả là 160

9. Toán tử điều kiện

Ngôn ngữ lập trình C đưa thêm vào một toán tử mới có dạng cú pháp như sau:

E1?E2:E3 ;

Trong đó toán tử *?:* được gọi là toán tử điều kiện. Còn ***E1***, ***E2*** và ***E3*** là các biểu thức nào đó. Sự kết hợp của toán tử điều kiện và các biểu thức *E1*, *E2* và *E3* tạo thành một biểu thức mới có tên là *biểu thức với toán tử điều kiện* [20]. Giá trị của biểu thức này sẽ bằng giá trị của *E2* nếu *E1* có giá trị khác không (*E1 đúng*), hoặc bằng giá trị của *E3* nếu giá trị của *E1* bằng không (*E1 sai*). Kiểu của biểu thức với toán tử điều kiện là kiểu cao nhất trong các kiểu của *E2* và *E3* (*theo sơ đồ ở mục 2.2.1. phần 8*).

Chú ý:

- *E1* không nhất thiết phải là một biểu thức quan hệ và logic (*chỉ có giá trị 0=sai và 1=đúng*) mà có thể là một biểu thức bất kì.

- Biểu thức với toán tử điều kiện rất tiện lợi trong việc tạo ra các ***Macro*** làm cho chương trình mềm dẻo hơn.

Ví dụ 2-8. Để tìm số lớn nhất trong hai số ta có thể viết như sau:

```
/* ************************************************************************************* */
#include "stdio.h"
#include "conio.h"
#define Max(A,B)  (A)>(B)?(A):(B)
/* ************************************************************************************* */
int main()
{
    int x=10, y=65;
    long a=3478925, b=2345677;
    float c=2.4, d=4.7;
    printf("\nSo lon nhat trong hai so x, y la: %10d", Max(x,y));
    printf("\nSo lon nhat trong hai so a, b la: %10ld", Max(a,b));
    printf("\nSo lon nhat trong hai so c, d la: %10.2f", Max(c,d));
    getch();
    return 0;

}
```

[20] Trong một số sách thường gọi đây là biểu thức điều kiện. Tuy nhiên cách gọi này có thể gây cho người học, đặc biệt là những người đã học qua ngôn ngữ Pascal, một sự nhầm lẫn với các biểu thức điều kiện được sử dụng trong các toán tử điều khiển như if… else, while…

Kết quả chạy chương trình như sau:

```
Số lon nhat trong hai so x, y la:          65
Số lon nhat trong hai so a, b la:     3478925
Số lon nhat trong hai so c, d la:        4.70
/* ******************************************************************************** */
```

Ta nhận thấy rằng, nếu ta viết chương trình này dưới dạng hàm, thì ta cần phải có 3 hàm khác nhau (*cho 3 kiểu dữ liệu khác nhau là int, float và long*) làm cùng một nhiệm vụ là tìm ra số có giá trị lớn hơn trong hai số. Điều này thật bất tiện cho việc lập trình.

Ví dụ 2-9. Trong ngôn ngữ lập trình C, để tìm giá trị tuyệt đối của một số ta có 3 hàm khác nhau có tên như sau:

Để tìm giá trị tuyệt đối của một số nguyên **x** ta dùng hàm :
int abs(int x), hàm này được khai báo trong thư viện **stdlib.h**
Để tìm giá trị tuyệt đối của một số thực **y** ta dùng hàm:
double fabs(double y), hàm này lại được khai báo trong **math.h**
Để tìm giá trị tuyệt đối của một số nguyên dài **b** ta lại dùng hàm:
long labs(long b), hàm này được khai báo trong **stdlib.h**

Đây quả thực là một điều hết sức phi lý, vì cùng một chức năng ta lại phải dùng đến 3 hàm khác nhau, được khai báo trong các thư viện khác nhau. Bằng cách viết *Macro*, ta có thể viết lại hàm tính trị tuyệt đối như sau cho các kiểu dữ liệu:

```
/* ******************************************************************************** */
#include "stdio.h"
#include "conio.h"
#define Abs(A)  (A)>=0?(A):-(A)
/* ******************************************************************************** */
int main()
{
    int a=-10;
    long b=1234567;
    float c=- 6.7;
    printf("\nTri tuyet đoi cua so a la: %10d",Abs(a));
    printf("\nTri tuyet đoi cua so b la: %10ld",Abs(b));
    printf("\nTri tuyet đoi cua so c la: %10.2f",Abs(c));
    getch();
    return 0;
}
/* ******************************************************************************** */
```

Kết quả chạy chương trình:

```
Tri tuyet đoi cua so a la:        10
Tri tuyet đoi cua so b la:   1234567
Tri tuyet đoi cua so c la:      6.70
/* ******************************************************************************** */
```

Việc dùng toán tử điều kiện để viết các *Macro* tuy có nhiều ưu điểm nhưng chỉ nên áp dụng cho các thao tác tương đối đơn giản. Đối với các thao tác phức tạp, ta vẫn nên viết chương trình dưới dạng các *hàm* (*chương 4 sẽ đề cập đến vấn đề này*) để giảm kích thước của chương trình, tuy nhiên trong trường hợp này tốc độ của chương trình sẽ bị chậm đi (*việc lí giải được xem như bài tập*).

10. Thứ tự ưu tiên của các phép toán

Thứ tự ưu tiên của các phép toán là một vấn đề hết sức quan trọng, trong biểu thức bao giờ các phép toán có thứ tự ưu tiên cao hơn cũng được thực hiện trước và ngược lại. Bảng 2.6. sẽ tổng kết thứ tự ưu tiên của các phép toán trong ngôn ngữ C.

Bảng 2.6. Thứ tự ưu tiên của các phép toán trong ngôn ngữ lập trình C

Mức ưu tiên	Các toán tử	Thứ tự kết hợp
1	() [] -> .	Trái qua phải
2	! ~ - (phép đổi dấu) ++ -- & (dùng để lấy địa chỉ của biến) * (dùng để khai báo con trỏ xét trong chương 3) () (phép ép kiểu) và **sizeof**	Phải qua trái
3	* (phép nhân) / %	Trái qua phải
4	+ - (phép trừ)	Trái qua phải
5	<< >>	Trái qua phải
6	< <= > >=	Trái qua phải
7	== !=	Trái qua phải
8	& (phép thao tác bit AND)	Trái qua phải
9	^	Trái qua phải
10	|	Trái qua phải
11	&&	Trái qua phải
12	||	Trái qua phải
13	?:	Phải qua trái
14	= += -= *= /= %= <<= >>= &= ^= |=	Phải qua trái
15	,	Trái qua phải

Trong đó, các phép toán trên một dòng có cùng thứ tự ưu tiên, tuy nhiên trật tự kết hợp của chúng lại khác nhau (*có thể từ trái qua phải hoặc ngược lại*). Các phép toán ở dòng trên có thứ tự ưu tiên cao hơn các phép toán ở dòng dưới. Mức ưu tiên cao nhất là các phép toán: '()' dùng để viết các biểu thức, '[]' dùng để truy nhập các phần tử mảng, '->' dùng để truy nhập thành phần cấu trúc thông qua con trỏ (*xem chương 3*) và phép '.' dùng để truy nhập các thành phần cấu trúc thông qua biến. Mức ưu tiên thấp nhất là toán tử ',' dùng để phân tách các biến khi khai báo, phân tách danh sách đối trong các hàm...

Chú ý:

Nếu không nhớ thứ tự ưu tiên của các phép toán, ta nên sử dụng dấu ngoặc tròn '*()*' để viết các biểu thức.

Ví dụ 2-10. Khi viết biểu thức: a>b?a:x?y:z
Ta có thể viết rõ hơn như sau: a>b?a:(x?y:z)

11. Một số ví dụ minh hoạ

Ví dụ 2-11. Hãy viết chương trình tính giá trị của biểu thức sau rồi đưa kết quả ra màn hình: $(\frac{x+y}{20} + 3.x^5)10+5.x^4-4.x^3+12.x^2-6.x+93$ với *x*, *y* là các giá trị nhập từ bàn phím.

Giải: Trong ngôn ngữ lập trình C, để tính được y^x ta sử dụng hàm:
double pow(double y, double x) khai báo trong *math.h* (*xem phụ lục III để biết thêm các hàm khác của Turbo C*).

```
/* ******************************************************************************* */
#include "stdio.h"
#include "conio.h"
#include "math.h"
/* ******************************************************************************* */
int main()
{
     float x,y;
     double z;
     printf("\nHay nhap gia tri cho cac so x, y:\n");
     scanf("%f%f",&x, &y);
     z=(((x+y)/20+3*pow(x, 5))*10+5*pow(x, 4)-4*pow(x, 3)+12*pow(x, 2)-6*x+93);
     printf("\nGia trị cua bieu thuc can tinh la: %10.4f",z);
     getch();
     return 0;
}
/* ******************************************************************************* */
```

Kết quả chạy chương trình:

```
     Hay nhap gia trị cho cac so x, y:
     3 2
     Gia tri cua bieu thuc can tinh la:    7772.5000
/* ******************************************************************************* */
```

Nhận xét:

Chương trình trên có sử dụng hàm ***pow()*** để tính các biểu thức dạng luỹ thừa. Tuy nhiên hàm này sử dụng các đối là các biến kiểu ***double*** và giá trị trả về cũng là kiểu ***double***. Do đó, trong chương trình trên khi ta khai báo các biến x,y là các biến kiểu ***float*** thì chương trình sẽ tự động chuyển các giá trị đó sang kiểu ***double***. Kết quả cuối cùng của biểu thức cũng có kiểu là ***double***.

Ví dụ 2-12. Lập chương trình đọc từ bàn phím độ dài ba cạnh của tam giác *ABC* rồi tính diện tích và các đường cao của tam giác đó (*chương trình không cần kiểm tra điều kiện để hình thành tam giác*).

Giải: Để tính được diện tích của tam giác ta sử dụng công thức *Hêrông* sau:

$S = \sqrt{P(P-a)(P-b)(P-c)}$ trong đó P là nửa chu vi và a, b, c là độ dài tương ứng với các cạnh của tam giác. Để tính được căn bậc hai của một số x nào đó ta sử dụng hàm *double sqrt(double x)* khai báo trong *math.h*.

```
/* ******************************************************************************* */
#include "stdio.h"
#include "conio.h"
#include "math.h"
/* ******************************************************************************* */
int main()
{
     float a, b, c, P;
     double Ha, Hb, Hc, S;
     printf("\nChuong trinh tinh dien tich va duong cao cua tam giac");
     printf("\nHay nhap đo dai ba canh a, b, c:\n");
```

```
    scanf("%f%f%f", &a, &b, &c);
    P=(a+b+c)/2;
    S=sqrt(P*(P-a)*(P-b)*(P-c));
    Ha=2*S/a;
    Hb=2*S/b;
    Hc=2*S/c;
    printf("\nDien tich cua tam giac ABC co cac canh\na= 10.2f\nb=%10.2f\
    \nc= %10.2f \nla:  %10.2f", a, b, c, S);
    printf("\nCac duong cao tuong ung la:\nHa= %10.2f\nHb= %10.2f \nHc=\
    %10.2f", Ha, Hb, Hc);
    getch();
    return 0;
}
/* ******************************************************************************** */
```

***Kết quả chạy chương trình*:**

```
    Chuong trinh tinh dien tich va duong cao cua tam giac
    Hay nhap đo dai ba canh a, b, c:
    7  4  5
    Dien tich cua tam giac ABC co cac canh
    a=          7.00
    b=          4.00
    c=          5.00
    là:         9.80
    Cac duong cao tuong ung la:
    Ha=       2.80
    Hb=       4.90
    Hc=       3.92
/* ******************************************************************************** */
```

Ví dụ 2-13. Lập chương trình tính và in giá trị biểu thức sau đây ra màn hình:

$Y = \dfrac{3\sin^2(2x+1) + 2\cos(3x+2)}{2}$ với x là một số thực nhập từ bàn phím.

Giải: Để tính được giá trị của biểu thức ta cần sử dụng các hàm thư viện ***double cos(double x)*** và ***double sin(double x)*** được khai báo trong *math.h*.

```
/* ******************************************************************************** */
#include "stdio.h"
#include "conio.h"
#include "math.h"
/* ******************************************************************************** */
int main()
{
    float x;
    double Y;
    printf("\nHay nhap gia tri cua x:\n");
    scanf("%f",&x);
    Y=(3*pow(sin(2*x+1),2)+2*cos(3*x+2))/2;
    printf("\nGia tri cua bieu thuc la: %10.4f",Y);
    getch();
    return 0;
}
```

/* ** */

Kết quả chạy chương trình:

```
Hay nhap gia tri cua x:
20.3
Gia tri cua bieu thuc la:    1.7085
```

/* ** */

2.2.2. Các toán tử điều khiển

Trong lập trình có cấu trúc, mỗi chương trình bao gồm nhiều câu lệnh được thực hiện một cách tuần tự theo thứ tự mà chúng được viết (*cấu trúc tuần tự*). Tuy nhiên trong hầu hết các trường hợp ta cần điều khiển thứ tự thực hiện các câu lệnh theo một trật tự nào đó để có thể giải quyết được các vấn đề đặt ra. Công cụ giúp cho lập trình viên có thể làm được điều này chính là các toán tử điều khiển. Các toán tử điều khiển trong ngôn ngữ lập trình C thực chất là các lệnh của ngôn ngữ dùng để tổ chức ra các loại cấu trúc trong chương trình (*cấu trúc lặp, rẽ nhánh*). Về mặt công dụng có thể chia các toán tử điều khiển thành *4* nhóm chính:

- Lệnh nhảy không điều kiện
- Nhóm các lệnh rẽ nhánh
- Nhóm các lệnh dùng để tổ chức chu trình
- Nhóm các lệnh điều khiển khác.

1. Lệnh nhảy không điều kiện

Là câu lệnh dùng để bẻ gãy tính tuần tự của một chương trình viết bằng ngôn ngữ lập trình C, nó có cú pháp như sau:

```
goto Nhan;
```

Trong đó, *Nhan* là một tên có dấu ':' đứng sau dùng để gán cho bất kì một câu lệnh nào trong chương trình. Ví dụ: Tiep: ++i; thì *Tiep* là *nhãn* của của câu lệnh *++i*.

Khi gặp toán tử này máy sẽ nhảy tới câu lệnh có nhãn viết sau từ khóa *goto* bỏ qua các câu lệnh đứng trước hoặc đứng sau lệnh nhảy này.

Chú ý:

- Câu lệnh *goto* và nhãn *Nhan* phải nằm trong cùng một hàm (*không thể dùng để nhảy từ hàm này sang hàm khác*).

- Không thể dùng toán tử *goto* để nhảy từ ngoài vào trong một khối lệnh. Nhưng việc nhảy từ trong ra ngoài một khối lệnh lại hoàn toàn hợp lệ.

- Trong chương trình hạn chế dùng toán tử *goto* vì nó phá vỡ tính cấu trúc của chương trình.

2. Nhóm các lệnh rẽ nhánh

Là nhóm các lệnh dùng để ra các quyết định rẽ nhánh, nhảy tới thực hiện một câu lệnh (*khối lệnh*) nào đó trong chương trình dựa vào giá trị thực tế của một biểu thức nào đó.

a) Toán tử if

Là *toán tử quyết định hai lựa chọn* tùy thuộc vào giá trị bằng không (*sai*) hay khác không (*đúng*) của *biểu thức* và có cú pháp như sau:

Dạng 1
if (Biểu thức)
Câu_lệnh;

Dạng 2
if(Biểu thức)
Câu_lệnh1;
else
Câu_lệnh2;

Chú ý:

- ***Biểu thức*** có thể là biểu thức bất kì (Biểu thức nguyên, thực, quan hệ và logic, ...) miễn sao cứ trả về hoặc giá trị bằng không (*ứng với trường hợp sai*) hoặc giá trị khác không (*ứng với trường hợp đúng*).

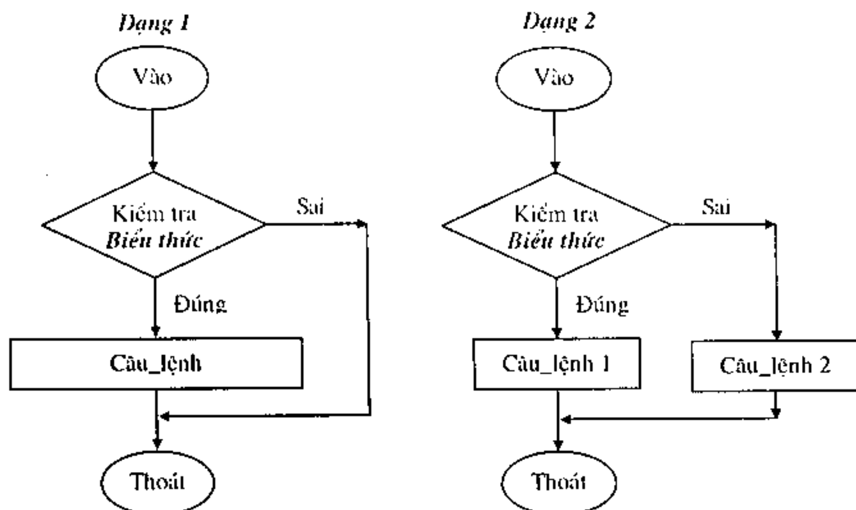
- Tại vị trí của ***Câu_lệnh*** nếu cần thực hiện nhiều câu lệnh ta phải đưa chúng vào trong khối lệnh.

- Khi viết chương trình, để tiện cho việc gỡ rối ta nên viết sao cho các câu lệnh và khối lệnh cùng cấp nên nằm trên một cột (*thẳng cột*), điểm đầu và điểm cuối của một khối lệnh cũng nên thẳng cột.

- ***Biểu thức*** bao giờ cũng phải được đặt trong hai dấu '*()*'.

- Trước từ khóa ***else*** vẫn phải có dấu ';'

Hoạt động của toán tử *if* có thể được mô tả bằng sơ đồ sau:



Hình 2.1. Sơ đồ hoạt động của toán tử if.

- Các toán tử ***if*** có thể lồng nhau, tuy nhiên khi số từ khóa ***else*** ít hơn số từ khóa ***if*** thì từ khóa ***else*** sẽ được gắn với ***if*** gần nhất trước đó.

Ví dụ 2-14. Với đoạn chương trình sau:

```
if (a>0)
        if (b>0)
                c=2003;
        else
                c=1999;
```

Thì ***else*** sẽ được gắn với từ khóa ***if*** bên dưới. Để tránh nhầm lẫn trong khi sử dụng các toán tử ***if*** lồng nhau ta nên sử dụng khối lệnh để xác định phạm vi cho từng toán tử ***if*** trong chương trình. Ví dụ trên có thể viết lại như sau:

```
if(a>0)
{
        if(b>0)
                c=2003;
        else
                c=1999;
}
```

Ví dụ 2-15. Lập chương trình để máy tính nói chuyện với người. Máy "nói" bằng các câu hiện trên màn hình, người "nói" bằng cách gõ từ bàn phím. Nội dung cuộc trò chuyện như sau:

Máy hỏi bạn tên, giới tính, có gia đình chưa? Nếu chưa thì máy khuyên bạn "Hãy can đảm và thận trọng! Chúc bạn (tên) nhiều may mắn !". Nếu bạn có rồi thì máy hỏi xem có mấy con. Nếu nhiều hơn hai con thì máy nói "Có vẻ hơi nhiều rồi đấy". Nếu trên năm con thì máy nói "Quá nhiều rồi anh (cô) bạn của tôi ơi !".

Giải: Để có thể viết được chương trình này chúng ta cần sử dụng các hàm xử lí chuỗi được khai báo trong thư viện ***string.h*** và ***stdlib.h*** của Turbo C sau đây (*các hàm khác có thể xem thêm trong phụ lục III*):

int strcmpi(char *s1, char *s2) dùng để so sánh hai chuỗi, không phân biệt chữ hoa và chữ thường, hàm cho:

- *Giá trị âm nếu chuỗi s1 nhỏ hơn chuỗi s2*
- *Giá trị không nếu chuỗi s1 bằng chuỗi s2*
- *Giá trị dương nếu chuỗi s1 lớn hơn chuỗi s2*

int atoi(char *s) dùng để chuyển chuỗi *s* sang giá trị nguyên. Hàm này khai báo trong ***stdlib.h***.

```
/* ************************************************************************************** */
#include "stdio.h"
#include "conio.h"
#include "string.h"
#include "stdlib.h"
/* ************************************************************************************** */
int main()
{     char GioiTinh[3], Ten[25], TraLoi[20];
      printf("\nXin chao, ban ten la gi ?\n");
      gets(Ten);
      printf("\nChao ban %s, ban la nam hay nu(Nam/Nu) ?\n", Ten);
      gets(GioiTinh);
HoiLai:
      printf("\nBan co gia dinh chua ?\n");
      gets(TraLoi);
      if (strcmpi(TraLoi,"Roi")==0)
      {     printf("\nBan da co may con roi ?\n");
            scanf("%s",TraLoi); /* nhập dưới dạng chuỗi kí tự  rồi chuyển thành số*/
            if(atoi(TraLoi)>=5)
            {   if(strcmpi(GioiTinh,"Nam")==0)
                        printf("\nQua nhieu roi anh ban %s cua toi oi !", Ten);
                else
                        printf("\nQua nhieu roi co ban %s cua toi oi !", Ten);
```

```
        }
        else  if (atoi(TraLoi)>=3)
                printf("\nCo ve hoi nhieu roi day !");
    }
    else
    {     if (strcmpi(TraLoi,"Chua")==0)
              printf("\nBan nen can dam va than trong! Chuc  %s may man hon!",
              Ten);
          else
              {   printf("\nBan nen go lai hoac \"Roi\" hoac \"Chua\"");
                   goto HoiLai;
              }
    }
    getch();
    return 0;
}
/* ************************************************************************************ */
```

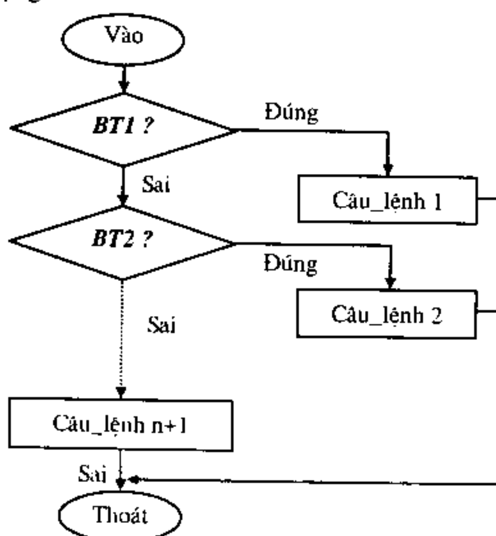
Kết quả chạy chương trình:

```
    Xin chao, ban ten la gi ?
    Binh
    Chao ban Binh, ban la nam hay nu (Nam/Nu) ?
    Nam
    Ban có gia dinh chua ?
    Roi
    Ban co may con roi ?
    5
    Qua nhieu roi anh ban Binh cua toi oi!
/* ************************************************************************************ */
```

b) Toán tử if mở rộng cho n lựa chọn

Cú pháp và nguyên tắc hoạt động như sau:

```
if(Biểu thức 1)
        Câu_lệnh1;
else if(Biểu thức 2)
        Câu_lệnh2;
else if(Biểu thức 3)
        Câu_lệnh3;
...
else if(Biểu thức n)
        Câu_lệnh n;
else
        Câu_lệnh n+1;
```



Hình 2.2. Sơ đồ hoạt động của toán tử if mở rộng cho n trường hợp

Chú ý:

- Tại vị trí các ***Câu_lệnh i*** nếu ta muốn thực hiện nhiều lệnh thì phải đặt chúng trong khối lệnh.

- Các ***Biểu thức*** được sử dụng cũng có thể là một biểu thức bất kì có giá trị bằng không (*ứng với trường hợp sai*) hoặc khác không (*ứng với trường hợp đúng*).

- Trong số ***n+1*** lựa chọn chỉ có một lựa chọn duy nhất mà thỏa mãn một ***Biểu thức i*** nào đó là được thực hiện. Nếu tất cả ***n Biểu thức*** đều không thỏa mãn thì ***Câu_lệnh n+1*** được thực hiện và câu lệnh này cũng có thể vắng mặt.

Ví dụ 2-16. Viết chương trình giao tiếp với máy tính, máy tính hỏi giới tính, tuổi của người giao tiếp. Nếu là nam thì máy sẽ nói „Chào ông" nếu tuổi >=90; máy nói "Chào Bác" nếu tuổi >=60; máy nói „Chào anh" nếu tuổi >=30; máy nói „Chào bạn" nếu tuổi <30. Nếu là nữ thì máy sẽ nói „Chào bà" nếu tuổi >=80; máy nói "Chào Bác" nếu tuổi >=50; máy nói „Chào chị" nếu tuổi >=25; máy nói „Chào em" trong các trường hợp còn lại. Chương trình kiểm tra cả trường hợp người nhập gõ nhầm kí tự (*không phải Nam cũng không phải Nu*).

```
/* ****************************************************************************************** */
#include "stdio.h"
#include "conio.h"
#include "string.h"
/* ****************************************************************************************** */
int main()
{
     char GioiTinh[3];
     int Tuoi;
HoiLai:
     printf("\nChao ban, ban la nam hay nu(Nam/Nu) ?\n");
     scanf("%s", GioiTinh);
     if((strcmpi(GioiTinh, "Nam")!=0)&&(strcmpi(GioiTinh, "Nu")!=0))
     {
           printf("\nHay go\" Nam\" hoac\" Nu\"");
           goto HoiLai;
     }
     printf("\nBan nam nay bao nhieu tuoi ?\n");
     scanf("%d", &Tuoi);
     if(strcmpi(GioiTinh, "Nam")==0)
       {
             if(Tuoi>=90)
                  printf("\nChao ong!");
             else if(Tuoi>=60)
                  printf("\nChao bac!");
             else if(Tuoi>=30)
                  printf("\nChao anh!");
             else
                  printf("\nChao ban!");
       }
     else
       {
```

```
            if(Tuoi>=80)
                printf("\nChao ba!");
            else if(Tuoi>=50)
                printf("\nChao bac!");
            else if(Tuoi>=25)
                printf("\nChao chi!");
            else
                printf("\nChao em!");
        }
    getch();
    return 0;
}
/* ****************************************************************************** */
```

Kết quả chạy chương trình:

```
    Chao ban, ban la nam hay nu (Nam/Nu) ?
    Nu
    Ban nam nay bao nhieu tuoi ?
    100
    Chao ba!
/* ****************************************************************************** */
```

c) Toán tử switch

Toán tử ***switch*** làm việc giống như toán tử ***if*** mở rộng cho *n* trường hợp ở trên, chỉ khác một điều là ***switch*** luôn làm việc với các ***biểu thức nguyên***.

Cú pháp của toán tử switch như sau:

```
switch (Biểu thức nguyên)
{
        case n1:
                Các câu lệnh;
                break;
        case n2:
                Các câu lệnh;
                break;
        ...
        case nk:
                Các câu lệnh;
                break;
        [default:
                Các câu lệnh;]
}
```

Trong đó, ***ni*** là các *số nguyên, hằng kí tự* hoặc *biểu thức hằng*. Các *ni* phải có giá trị khác nhau. Các câu lệnh nằm trong hai dấu '*{}*' được gọi là thân của ***switch***. Ta có thể mô tả hoạt động của ***switch*** như sau:

- Trước tiên, máy sẽ xét giá trị của ***Biểu thức nguyên***, tùy theo giá trị của biểu thức này mà nó quyết định nhảy tới đâu.

- Nếu giá trị này bằng ***ni***, máy sẽ nhảy tới câu lệnh nằm sau nhãn ***case ni*** và bắt đầu thực hiện các lệnh từ đó cho đến khi gặp một toán tử ***break***, ***goto***, ***return*** hoặc dấu kết thúc '*}*'.

- Khi giá trị của biểu thức nguyên khác tất cả các *ni*, *i=1÷k* thì sự hoạt động của *switch* lúc này phụ thuộc vào sự có mặt hay vắng mặt của ***default***. Nếu có mặt ***default*** thì các câu lệnh nằm sau ***default*** sẽ được thực hiện, ngược lại máy sẽ thoát khỏi *switch* và bắt đầu thực hiện các lệnh nằm sau dấu '*}*' của thân *switch*.

Chú ý:

- Máy sẽ thoát khỏi toán tử ***switch*** chỉ khi nó gặp một toán tử ***break*** hoặc dấu ngoặc đóng '*}*' chỉ sự kết thúc của toán tử ***switch***. Do đó các toán tử ***break*** là không thể vắng mặt mỗi khi một nhánh nào đó đã được lựa chọn.

- Trong thân của lệnh ***switch*** ta cũng có thể sử dụng toán tử ***goto*** để nhảy tới một câu lệnh bất kì bên ngoài ***switch***.

- Khi ***switch*** nằm trong thân của một hàm nào đó, ta cũng có thể sử dụng một toán tử ***return*** trong thân của ***switch*** để thoát khỏi hàm đó (*xem thêm 2.2.2. mục 4*).

Ví dụ 2-17. Viết chương trình mô phỏng việc thực hiện bốn phép tính cơ bản cộng, trừ, nhân và chia với các toán hạng và toán tử nhập từ bàn phím.

Ví dụ nhập 2+3 sẽ cho kết quả là 5...

Giải: trong chương trình cần sử dụng đến hàm ***fflush(stdin)*** được khai báo trong ***stdio.h*** để làm sạch bộ đệm bàn phím trước khi nhập kí tự.

```
/* ************************************************************************************************ */
#include "stdio.h"
#include "conio.h"
/* ************************************************************************************************ */
int main()
{   float num1, num2;
    char KyTu;
LapLai:
    printf("\nHay nhap bieu thuc duoi dang\"So ToanTu So\":\n");
    scanf("%f%*[ \n]", &num1);
    scanf("%c%f", &KyTu, &num2);
    switch(KyTu)
    {
        case '+':
            printf(" = %.2f", num1+num2);
            break;
        case '-':
            printf(" = %.2f", num1-num2);
            break;
        case '*': /* người sử dụng có thể gõ axb hoặc aXb hoặc a*b */
        case 'x':
        case 'X':
            printf(" = %.2f", num1*num2);
            break;
        case '/': /* người sử dụng có thể gõ a/b hoặc a\b đều được */
```

```
        case '\\':
              printf(" = %.2f", num1/num2);
              break;
        default:
              printf("\n Toan tu khong quen biet !");
    }
    printf("\nBan co muon tinh tiep khong?(C/K)");
    fflush(stdin);
    scanf("%c",&KyTu);
    if(KyTu=='C'||KyTu=='c')
        goto LapLai;
    getch();
    return 0;
}
/* ***************************************************************************************** */
```

Kết quả chạy chương trình:

```
    Hay nhap bieu thuc duoi dang "So  ToanTu So":
    23 +  60= 83.00
    Ban co muon tinh tiep khong ?(C/K)
    c
    Hay nhap bieu thuc duoi dang "So  ToanTu So":
    12 x  20= 240.00
    Ban co muon tinh tiep khong ?(C/K)
    K
/* ***************************************************************************************** */
```

Vấn đề cần giải đáp:

- Chương trình trên sử dụng định dạng ***%*[\n]*** trong câu lệnh *scanf* để làm gì ? hãy giải thích!

- Giả sử không sử dụng hàm ***fflush(stdin)*** thì vấn đề gì sẽ xảy ra ? Tại sao ?

3. Nhóm các lệnh dùng để tổ chức chu trình

Trong lập trình có cấu trúc, thông thường ta cần tạo ra các đoạn mã mà hoạt động của nó lặp đi lặp lại theo một quy luật nào đó (*như đoạn mã dùng cấu trúc goto và if trong ví dụ 2-17 chẳng hạn*). Để thuận tiện cho việc tạo ra các chu trình trong khi viết chương trình, Turbo C đưa ra hai loại cấu trúc lặp với mục đích khác nhau, đó là cấu trúc lặp xác định (*dùng toán tử điều khiển **for***) và cấu trúc lặp không xác định (*dùng toán tử điều khiển **while** và **do … while***). Cấu trúc lặp xác định (*hay còn gọi là vòng lặp xác định*) thường hay dùng cho các thao tác lặp đi lặp lại với *số lần lặp đã được biết trước*. Còn cấu trúc lặp không xác định (*hay còn gọi là vòng lặp không xác định*) thường hay dùng trong các trường hợp ta chưa biết trước sẽ phải lặp bao nhiêu lần, số lần lặp này thường phụ thuộc vào giá trị tại thời điểm hiện tại của một hay nhiều biểu thức nào đó.

a) Toán tử điều khiển for

Là toán tử dùng để xây dựng các ***vòng lặp xác định*** trong chương trình. Trong ngôn ngữ lập trình C, cấu trúc của vòng lặp ***for*** linh động, hiệu quả hơn trong *Pascal*, cú pháp của nó như sau:

```
for(Biểu thức 1; Biểu thức 2; Biểu thức 3)
```

Câu_lệnh; /*thân của vòng lặp*/

Hoạt động của vòng lặp for:

B1: *Thực hiện Biểu thức 1 rồi chuyển sang B2.*

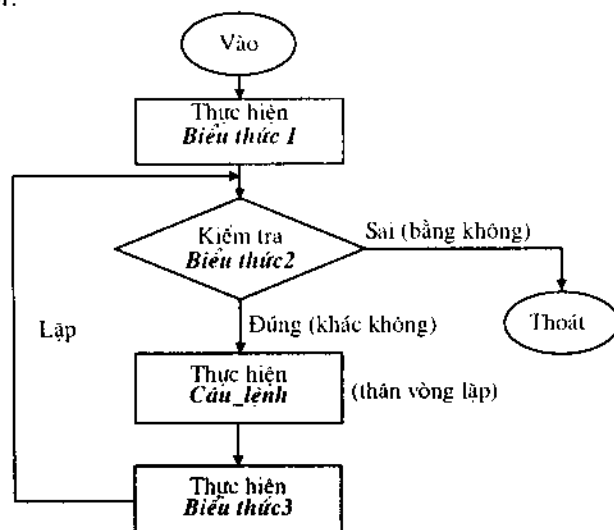
B2: *Kiểm tra Biểu thức 2:*

- Nếu Biểu thức 2 đúng (có giá trị khác không) thì thực hiện câu lệnh hoặc khối lệnh trong thân vòng for rồi chuyển sang B3.

- Nếu Biểu thức 2 bằng không (sai) thì chuyển sang B4.

B3: *Thực hiện Biểu thức 3 rồi quay lại B2.*

B4: *Thoát.*



Hình 2.2. Sơ đồ hoạt động của toán tử for.

Ví dụ 2-18. Viết chương trình phát ra 20 tiếng ***bíp***.

Giải: Để có thể phát ra các tiếng bíp ta dùng hàm ***printf*** thực hiện chức năng của kí tự ***Bell*** (*mã ASCII của nó là 7*). Tuy nhiên, sau khi phát ra một tiếng bip ta cần tạm dừng một khoảng thời gian nhất định (*nếu không ta chỉ nghe thấy dường như một tiếng bip dài mà thôi*) bằng hàm:

void delay(int n) được khai báo trong thư viện ***dos.h***. Hàm này sẽ thực hiện tạm dừng hoạt động của máy trong khoảng ***n*** miligiây.

```
/* ******************************************************************************** */
#include "stdio.h"
#include "conio.h"
#include "dos.h"
/* ******************************************************************************** */
int main()
{
    int  i;
    printf("\nChương trình phát 20 tiếng bíp");
    for(i=1; i<=20; ++i)
    {
```

```
        printf("%c",7);
        delay(1000);
    }
    getch();
    return 0;
}
/* ******************************************************************************************* */
```

Chú ý:

- Trong thân toán tử ***for*** nếu muốn thực hiện nhiều câu lệnh thì phải đặt trong khối lệnh.

- Trong phần ngoặc tròn '()' của toán tử ***for*** được chia thành 3 phần phân cách nhau bởi dấu ';'. Các dấu ';' này là bắt buộc phải có mặt, kể cả trường hợp một hoặc một số *Biểu thức* bị vắng mặt. Trong mỗi phần lại có thể đặt nhiều hơn một biểu thức phân cách nhau bởi ***dấu phẩy*** ',' để thực hiện các công việc khác nhau theo thứ tự xác định từ trái qua phải. Khi phần chứa ***Biểu thức 2*** có nhiều hơn một Biểu thức thì tính *đúng sai* của toàn toán tử ***for*** sẽ là tính đúng sai của biểu thức cuối cùng trong dãy biểu thức của phần này. Ta có thể hiểu kĩ hơn hoạt động của vòng ***for*** thông qua ví dụ 2-19 dưới đây.

- Toán tử ***for*** cũng có thể sử dụng để xây dựng các vòng lặp không xác định bằng cách điều khiển hợp lí các biểu thức của nó.

Ví dụ 2-19. Hãy sử dụng toán tử ***for*** để xây dựng vòng lặp không xác định thay cho vai trò của toán tử ***goto*** và toán tử ***if*** trong ***ví dụ 2-17*** ở trên.

```
/* ******************************************************************************************* */
#include "stdio.h"
#include "conio.h"
/* ******************************************************************************************* */
int main()
{
    float num1, num2;
    char KyTu='C';
    for( ; KyTu=='c'||KyTu=='C'; printf("\nBan co muon tinh tiep khong?(C/K)"),
    scanf("%*[ \n]%c", &KyTu))
    {
        printf("\nHay nhap bieu thuc duoi dang \"So ToanTu So\":\n");
        scanf("%f%*[ \n]", &num1);
        scanf("%c%f", &KyTu, &num2);
        switch(KyTu)
        {
            case '+':
                printf(" = %.2f",num1+num2);
                break;
            case '-':
                printf(" = %.2f",num1-num2);
                break;
            case '*':
            case 'x':
            case 'X':
```

```
                    printf(" = %.2f",num1*num2);
                    break;
                case '/':
                case '\\':
                    printf(" = %.2f",num1/num2);
                    break;
                default:
                    printf("\n Toan tu khong quen biet !");
        }
    }
    getch();
    return 0;
}
/* ************************************************************************************ */
```

Kết quả chạy chương trình cũng cho kết quả tương tự:

```
    Hay nhap bieu thuc duoi dang "So  ToanTu So":
    23 +  60= 83.00
    Ban co muon tinh tiep khong ?(C/K)
    c
    Hay nhap bieu thuc duoi dang "So  ToanTu So":
    12 x  20= 240.00
    Ban co muon tinh tiep khong ?(C/K)
    K
/* ************************************************************************************ */
```

- ***Biểu thức 1*** bao giờ cũng được thực hiện đúng một lần.

- Khi ***Biểu thức 2*** vắng mặt thì nó luôn được xem là đúng. Để ra khỏi chu trình trong trường hợp này ta phải dùng đến các lệnh ***break***, ***goto*** hoặc ***return*** bên trong thân vòng lặp.

- Thân của vòng for có thể là một câu lệnh hoặc một khối lệnh.

- Bên trong thân của vòng ***for*** lại có thể chứa các toán tử ***for*** khác. Điều đó có nghĩa là các toán tử ***for*** có thể đặt lồng nhau.

Ví dụ 2-20. Viết chương trình in ra màn hình bàn cờ quốc tế (*kí tự đồ hoạ*).

```
/* ************************************************************************************ */
#include "stdio.h"
#include "conio.h"
/* ************************************************************************************ */
int main()
{
    int i, j;
    for(i=1; i<=8; i++)
    {
        for(j=1; j<=8; j++)
            if((i+j)%2==0)/*Xét các ô cùng màu*/
                printf("\xDB");/*Tô hình vuông trắng*/
            else
                printf(" "); /*Để trống->các ô đen (lấy theo màu nền)*/
```

```
        printf("\n");/*Xét dòng tiếp theo*/
    }
    getch();
    return 0;
}
/* *********************************************************************************** */
```

b) Toán tử điều khiển while

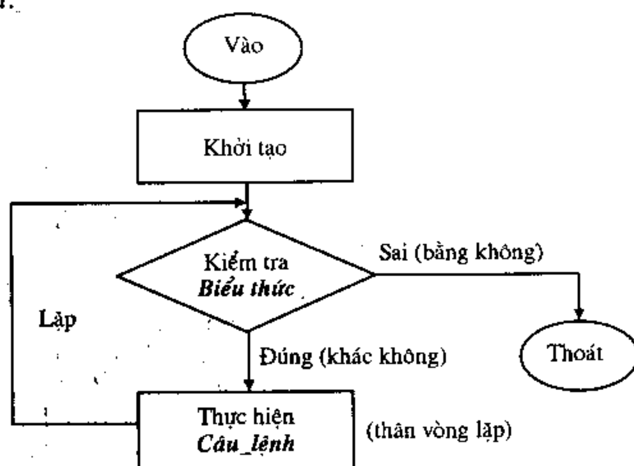
Là toán tử dùng để xây dựng các vòng lặp không xác định *kiểm tra điều kiện trước*. Cú pháp của nó như sau:

```
while(Biểu thức)
        Câu_lệnh; /*Thân vòng lặp while*/
```

Hoạt động của toán tử while:

B1: *Khởi tạo, chuẩn bị ban đầu.*
B2: *Kiểm tra Biểu thức:*
- *Nếu Biểu thức đúng (có giá trị khác không) thì chuyển sang B3.*
- *Nếu Biểu thức bằng không (sai) thì chuyển sang B4.*

B3: *Thực hiện câu lệnh hoặc khối lệnh trong thân vòng lặp rồi quay lại B2.*
B4: *Thoát.*



Hình 2.4. Sơ đồ hoạt động của toán tử while.

Chú ý:

- Trong thân vòng lặp *while* (*tại vị trí của **Câu_lệnh***) nếu muốn thực hiện nhiều câu lệnh thì phải đặt trong một khối lệnh.

- Thân vòng lặp có thể được thực hiện một hoặc nhiều lần và cũng có thể không được thực hiện lần nào nếu ngay từ ban đầu ***Biểu thức*** đã sai (*bằng không*).

- Cũng giống như toán tử ***for***, trong cặp ngoặc tròn '*()*' sau từ khóa ***while*** ta có thể đặt nhiều hơn một biểu thức phân cách nhau bởi dấu phẩy ','. Tuy nhiên tính đúng sai của dãy biểu thức sẽ là tính đúng sai của biểu thức sau cùng trong dãy.

- Bên trong thân của một toán tử ***while*** lại có thể là toán tử ***for*** hoặc ***while*** khác. Nói cách khác là ta có thể xây dựng được các chu trình lồng nhau.

- Phần khởi tạo là rất quan trọng nhằm tạo ra giá trị ban đầu cho ***Biểu thức*** trước khi vào vòng lặp ***while***.

- Nếu Biểu thức trong '*()*' của vòng lặp ***while*** luôn đúng, ta sẽ có một vòng lặp vô tận. Do đó trong khi viết vòng lặp, bao giờ ta cũng phải kiểm tra tính dừng của vòng lặp đó. Sau đây là một ví dụ vi phạm tính dừng của vòng lặp do sử dụng biến điều khiển một cách bừa bãi.

Ví dụ 2-21. Ví dụ sai vì sử dụng bừa bãi biến điều khiển.

```
/* ******************************************************************************** */
#include "stdio.h"
#include "conio.h"
/* ******************************************************************************** */
int main()
{
    int i=10;
    while(i!=0)
    {
        printf("\nGiá trị của i là: %d", i);
        for(i=1; i<2; i++)
            printf("%c",7);
        --i;
    }
    getch();
    return 0;
}
/* ******************************************************************************** */
```

Ví dụ 2-22. Viết chương trình cho dòng chữ Hello ! chạy ngang màn hình trong chế độ văn bản cho đến khi bấm phím ***Enter***.

Giải: Để có thể giải quyết được vấn đề này ta cần dùng thêm hàm kiểm tra bộ đệm bàn phím ***int kbhit(void)***, hàm nhận một kí tự từ bộ đệm bàn phím ***int getch(void)***, hàm xóa màn hình ***clrscr()*** và hàm di chuyển con trỏ màn hình đến một tọa độ (*x,y*) bất kì trong chế độ văn bản ***gotoxy(x,y)***. Các hàm này đều được khai báo trong thư viện ***conio.h***. Hàm ***kbhit*** sẽ cho giá trị khác không nếu có phím đã được bấm (*bộ đệm bàn phím khác rỗng*) ngược lại cho giá trị bằng không.

```
/* ******************************************************************************** */
#include "stdio.h"
#include "conio.h"
/* ******************************************************************************** */
int main()
{
    int i= 0;
    /* Làm cho bộ đệm trở nên rỗng trước khi bắt đầu chạy chữ */
    while(kbhit())
        getch();
    /* Dòng chữ sẽ chạy cho đến khi bấm phím Enter */
    while(1)
    {
        clrscr();
```

```
        /* Nếu dòng chữ chạy đến cuối màn hình thì quay trở lại */
        if (i==80)
            i=0;
        gotoxy( i++, 10);/* Làm cho dòng chữ chạy ngang*/
        printf("%s", "HELLO !");
        delay(10000);
        /* Kiểm tra xem phím Enter có bị bấm ? */
        if (kbhit())
            if(getch()=='\r')
                break;/* Đúng, thoát*/
    }
    return 0;
}
/* ******************************************************************************** */
```

c) Toán tử điều khiển do ... while

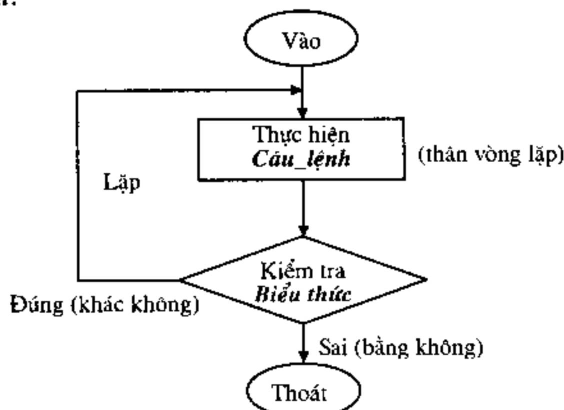
Trong các toán tử ***for*** và ***while*** đã xét ở trên, việc kiểm tra điều kiện kết thúc được tiến hành ở đầu của vòng lặp và do đó thân vòng lặp có thể sẽ không được thực hiện lần nào nếu ***biểu thức*** điều kiện ngay từ ban đầu đã không được thỏa mãn. Khác với hai toán tử này, toán tử ***do*** ... ***while*** lại kiểm tra điều kiện kết thúc ở cuối vòng lặp, cho nên thân của vòng lặp này bao giờ cũng được thực hiện ít nhất một lần (*giống vòng lặp **repeat until** trong Pascal*, tuy nhiên ***do*** ... ***while*** sẽ lặp trong khi điều kiện còn đúng, ngược lại ***repeat until*** lại lặp cho đến khi điều kiện đúng). Còn về cách thức làm việc thì ***do*** ... ***while*** làm việc tương tự như vòng lặp ***while***. Cú pháp của toán tử này như sau:

```
do
    Câu_lệnh;/*Thân của vòng lặp*/
while(Biểu thức);
```

Hoạt động của toán tử do while:

B1: *Thực hiện câu lệnh hoặc khối lệnh trong thân vòng lặp.*
B2: *Kiểm tra Biểu thức:*
- *Nếu Biểu thức đúng (có giá trị khác không) thì quay lại B1.*
- *Nếu Biểu thức bằng không (sai) thì chuyển sang B3.*

B3: *Thoát.*



Hình 2.5. Sơ đồ hoạt động của toán tử do ... while.

Chú ý:

- Các lưu ý của toán tử ***while*** cũng đúng cho toán tử ***do ... while***.

Ví dụ 2-23. Viết chương trình tính *căn bậc hai* của một số dương *a* thông qua công thức truy hồi sau:

$X_{n+1} = \frac{1}{2}\left(X_n + \frac{a}{X_n} \right)$; với n >=0, $X_0 = a$; Quá trình lặp kết thúc khi thỏa mãn hệ thức $\left|(X_{n+1} - X_n)/X_n\right| \le \varepsilon$, với ε nhập từ bàn phím. Khi đó giá trị của X_{n+1} được xem là giá trị gần đúng của căn bậc hai của *a* (*với sai số* ε). Trong chương trình ta cần dùng hàm ***double fabs(double)*** để trả về giá trị tuyệt đối của một số thực, hàm này khai báo trong ***math.h***.

```
/* ************************************************************************************** */
#include "stdio.h"
#include "conio.h"
#include "math.h"
/* ************************************************************************************** */
int main()
{
    double a, Epsilon, CanBacHai, Xcu, Xmoi;
    do
        {
            clrscr();
            printf("\nHay nhap so duong a va so duong EpSiLon \n");
            scanf("%lf%lf", &a, &Epsilon);
        }
    while((a<0)||(Epsilon<0)); /*Chỉ nhập các số dương*/
    Xmoi= a; /* Điều kiện đầu*/
    do
    {
        Xcu=Xmoi;
        Xmoi=(Xcu+ a/Xcu)/2;
    }
    while(fabs((Xmoi-Xcu)/Xcu)>Epsilon);
    printf("\nCan bac hai cua  %10.2f  la %10.4f", a, Xmoi);
    getch();
    return 0;
}
/* ************************************************************************************** */
```

4. Nhóm các lệnh điều khiển khác

a) Toán tử break

Là một toán tử điều khiển đặc biệt cho phép ra khỏi vòng lặp ***for***, ***while***, ***do while*** và toán tử ***switch*** trong trường hợp cần thiết khi đang thực hiện các câu lệnh trong thân vòng lặp hoặc thân của ***switch***. Điều đó có nghĩa là toán tử ***break*** cho ta khả năng ra khỏi một vòng lặp hoặc ***switch*** (*từ một điểm bất kì bên trong thân vòng lặp hoặc thân switch*) mà không dùng đến điều kiện kết thúc vòng lặp hoặc không muốn các lệnh sau ***break*** trong thân ***switch*** được thực hiện tiếp.

Chú ý:

- Khi có nhiều vòng lặp hoặc ***switch*** lồng nhau thì câu lệnh ***break*** chỉ thoát ra khỏi vòng lặp sâu nhất (*trong cùng*) có chứa toán tử ***break*** đó mà thôi.

- Mọi toán tử ***break*** đều có thể thay bằng toán tử ***goto*** với nhãn thích hợp.

- ***break*** thường được sử dụng khi vòng lặp ***for*** có thể thực hiện số lần lặp ít hơn số lần lặp đã được xác định trước, trong thân của switch và trong các trường hợp ta phải kiểm tra điều kiện kết thúc của vòng lặp ***while*** bên trong thân của nó như trong ví dụ 2-22 đã được trình bày ở trên.

Ví dụ 2-24. Viết chương trình xét xem một số nguyên dương *n* (*với n nhập từ bàn phím*) có phải là một số nguyên tố hay không ?

```
/* ******************************************************************************************** */
#include "stdio.h"
#include"conio.h"
#include "math.h"
/* ******************************************************************************************** */
int main()
{
    char ch;
    int i, Dung, n;
    /* Thực hiện chương trình cho đến khi bấm 'K' hoặc 'k'*/
    do
    {
        Dung=0; /* giả sử số đó là hợp số*/
        /* Chỉ nhập số nguyên dương */
        do
        {
            printf("\nHay nhap so nguyen duong n = ");
            scanf("%d", &n);
            if(n<0)
            {
                printf("%c", 7);
                printf("\nHay nhap lai");
            }
        }
        while(n<0);
        /* Xét tính nguyên tố của số vừa nhập */
        for(i=2; i<= sqrt(n); i++)
            if((n%i)==0)
                break; /* Thoát luôn */
        if (i>sqrt(n) && (n!=1) && (n!=0)) /* đã xét hết các ước số có thể có*/
             Dung=1; /* Chắc chắn là số nguyên tố */
        if(Dung)
            printf("\nSo %d la mot so nguyen to",n);
        else
            printf("\nSo %d la hop so",n);
        getch();
    }
```

```
    while( clrscr(), printf("\nBan co tiep tuc nua khong (C/K) ?"),
    scanf("%*[  \n]%c", &ch), ch!= 'K' && ch != 'k');
    return 0;
}
/* ************************************************************************************** */
```

b) Câu lệnh continue

Trái với toán tử ***break*** (*dùng để thoát ra khỏi vòng lặp*) câu lệnh ***continue*** dùng để bắt đầu một vòng lặp mới của chu trình sâu nhất chứa toán tử đó. Nói một cách khác:

- Khi gặp toán tử ***continue*** bên trong thân của một toán tử for, máy sẽ bỏ qua các câu lệnh còn lại trong thân ***for*** (*sau **continue***) chuyển sang thực hiện ***Biểu thức 3*** để khởi đầu cho vòng lặp mới tiếp theo;

- Khi gặp toán tử ***continue*** bên trong thân của vòng lặp ***while*** hoặc ***do while***, máy sẽ bỏ qua các lệnh còn lại trong thân vòng lặp (*sau **continue***) chuyển tới ***Kiểm tra Biểu thức*** sau từ khóa while để khởi động cho vòng lặp tiếp theo (*nếu Biểu thức còn đúng*) hoặc thoát khỏi vòng lặp (*nếu Biểu thức sai*).

Chú ý:

Toán tử ***continue*** chỉ áp dụng cho các vòng lặp mà thôi (*không áp dụng cho toán tử **switch***).

Ví dụ 2-25. Viết chương trình nhập các kí tự từ bàn phím sau đó chỉ cho hiện lên màn hình các kí tự là chữ hoa. Chương trình kết thúc khi bấm phím *ESC* (*mã ASCII của nó bằng 27*).

Giải: Để kiểm tra xem một kí tự nhận được có phải là chữ hoa hay không ta dùng hàm ***int isupper(int KyTu)*** được khai báo trong thư viện ***ctype.h*** của Turbo C. Hàm sẽ cho một giá trị khác không (*đúng*) nếu *KyTu* là một chữ hoa, ngược lại cho giá trị bằng không. Một điểm cần lưu ý nữa là ở đây ta không thể dùng hàm ***scanf*** để nhập phím được (*vì hàm scanf sẽ cho hiện tất cả các kí tự vừa nhập ra màn hình*) mà ta sử dụng hàm ***getch*** được khai báo trong ***conio.h***.

```
/* ************************************************************************************** */
#include "stdio.h"
#include "conio.h"
#include "ctype.h"
/* ************************************************************************************** */
int main()
{
    char KyTu;
    while(1)
    {
        KyTu=getch(); /*Nhập kí tự */
        if(!isupper(KyTu)) /*Là kí tự khác chữ hoa */
            if(KyTu==27) /* Điều kiện thoát*/
                break;
            else /* Chưa thoát, tiếp tục vòng sau*/
                continue;
```

```
        /* Là chữ hoa, được hiển thị ra màn hình*/
        printf("%c", KyTu);
    }
    return 0;
}
/* ************************************************************************************** */
```

c) Câu lệnh return

Là toán tử điều khiển đặc biệt dùng để thoát ra khỏi một hàm và trả về cho hàm một giá trị xác định nào đó nằm trong biểu thức sau từ khóa ***return*** (*trong chương 4 sẽ tìm hiểu sâu hơn về hàm trong ngôn ngữ lập trình C*). Cú pháp của nó như sau: ***return [Biểu thức]***;

Chú ý:

- Biểu thức sau ***return*** có thể vắng mặt (*trong trường hợp ta không cần trả về giá trị cho một hàm*).

- Khi gặp ***return*** máy sẽ thoát ra khỏi bất kì một vòng lặp nào (*kể cả trường hợp có nhiều vòng lặp lồng nhau*) và thoát luôn ra khỏi hàm có chứa các vòng lặp đó.

- Khi gặp một toán tử ***return*** có chứa ***Biểu thức***, thì giá trị của biểu thức sẽ được chuyển kiểu cho phù hợp với kiểu trả về của hàm trước khi nó được gán cho hàm.

5. Một số ví dụ minh hoạ

Ví dụ 2-26. Viết chương trình nhập vào một số nguyên N có giá trị từ 0 đến 9999. Hãy in ra bằng chữ giá trị của số đó (*ví dụ: 106 in ra là một trăm linh sáu*).

```
/* ************************************************************************************** */
#include"stdio.h"
#include"conio.h"
#define MAX  4
/* ************************************************************************************** */
main()
{
    int SoNguyen, i, k, j, ChuSo[MAX]; /* chứa các chữ số của số sẽ nhập*/
    printf("\nHay nhap mot so nguyen tu 0 den 9999\n");
    scanf("%d", &SoNguyen);
    ChuSo[0]= SoNguyen;
    for(i=MAX-1 ; i>0 ; --i)
    {
        ChuSo[i]=ChuSo[0] % 10;
        ChuSo[0] /= 10;
    }
    k= -1; /* điều chỉnh để k chỉ vào phần tử đầu tiên khác không trong mảng*/
    j=-1; /* điều chỉnh để j chỉ vào phần tử cuối cùng khác không trong mảng */
    for(i=0; i<MAX; ++i)
        if(ChuSo[i]!=0)
            break;
    if(i<MAX) k=i;
    for(i=MAX-1; i>=0; --i)
        if(ChuSo[i]!=0)
```

```
            break;
    if(i>=0) j= i;
    for( i=0; i< MAX ; ++i)
    {
        if(k== -1)
        {
            printf(" Khong"); break;
        }
        switch(ChuSo[i])
        {
            case 0:
                if ( (i>= k) &&(i<=j) ) /* là số 0 có nghĩa */
                {
                    if(i==2) /* là hàng chục*/
                    {
                        if(ChuSo[3]!=0) /* hàng đơn vị khác không*/
                            printf(" linh ");
                    }
                    else if(i==3) /* là hàng đơn vị*/
                    {
                        if(ChuSo[2]!=1) /* nếu khác số muoi*/
                            printf(" muoi ");
                    }
                    else  printf(" khong ");
                }
                break;
            case 1:
                if (i==2) /* hàng chục*/
                    printf(" muoi ");
                else   printf(" mot ");
                break;
            case 2:
                printf(" hai ");
                break;

            case 3:
                printf(" ba ");
                break;
            case 4:
                printf(" bon ");
                break;
            case 5:
                printf(" nam ");
                break;
            case 6:
                printf(" sau ");
                break;
            case 7:
                printf(" bay ");
                break;
            case 8:
```

```
                    printf(" tam ");
                    break;
                case 9:
                    printf(" chin ");
                    break;
        }
        switch(i)
        {
                case 0:
                    if(i>=k) /* chỉ hiện cho những số có nghĩa */
                            printf(" nghin ");
                    break;
                case 1:
                    if((i>=k)&&(i<=j)) /* chỉ hiện cho những số có nghĩa */
                            printf(" tram ");
                    break;
                case 2:
                    if(i>=k)
                        if((ChuSo[3]!=0)&&(ChuSo[2]!=0)&&(ChuSo[2]!=1))
                                printf(" muoi");
        }

    }
    getch();
}
/* ********************************************************************************************* */
```

Ví dụ 2-27. Viết chương trình tính và in ra tổng theo công thức sau:

$S = 2004 + 1 + \frac{1}{2!} + \frac{1}{3!} + ... + \frac{1}{n!}$ với n là một số nguyên dương bất kì.

Giải: Nhận xét rằng tổng trên được phân ra làm hai phần, một phần không có quy luật (*ta xem như giá trị ban đầu của tổng, đó là 2004*) và một phần có quy luật. Phần có quy luật có thể được tính bằng cách tính riêng từng phần tử của tổng rồi lần lượt cộng các phần tử đó vào tổng chung.

```
/* ********************************************************************************************* */
#include "stdio.h"
#include "conio.h"
/* ********************************************************************************************* */
int main()
{
    int i, n;
    double Tong, PhanTu_i;
    Tong=2004; /* giá trị ban đầu của tổng */
    PhanTu_i=1;
    /* Chỉ nhập số nguyên  dương n */
    do
    {
        printf("\nHay nhap so nguyen duong n= ");
        scanf("%d", &n);
        if(n<=0)
```

```
            printf("\nMoi ban nhap lai");
    }
    while(n<=0);
    /* Tính tổng đã cho */
    for(i=1; i<=n; i++)
    {
        PhanTu_i /=i;
        Tong += PhanTu_i;
    }
    printf("\nTong cua chuoi da cho voi n= %d la: %10.2f ", n, Tong);
    getch();
    return 0;
}
/* ************************************************************************************** */
```

Ví dụ 2-28. Viết chương trình xếp các dấu '*' thành tam giác cân n dòng, với n đọc từ bàn phím.

```
/* ************************************************************************************** */
#include "stdio.h"
#include "conio.h"
/* ************************************************************************************** */
int main()
{
    int n, i, j;
    clrscr();
    printf("\nCho biet so dong\n");
    scanf("%d", &n);
    clrscr();
    for(i=1; i<=n; i++)
    {
        gotoxy(40-i, i);
        for(j=1; j<=2*i-1; j++)
        {
            printf("*");
        }
        printf("\n");
    }
    gotoxy(10, 24);
    printf("An mot phim bat ky de ket thuc !");
    getch();
    return 0;
}
/* ************************************************************************************** */
```

Ví dụ 2-29. Viết chương trình tính và đưa ra màn hình các cặp giá trị x, $f(x)$ của hàm số $f(x) = \sqrt[5]{x^5 + \sqrt{|x|}}$ với x lấy dãy giá trị *-1; -0.9; -0.8; ...; 4.9; 5.*

```
/* ************************************************************************************** */
#include "stdio.h"
#include "conio.h"
#include "math.h"
/* ************************************************************************************** */
int main()
```

```
{
    int Dem;
    float x, f;
    clrscr();
    printf("%10s%10s\n", "x", "y=f(x)");
    x=-1.;
    Dem=0;
    while(x<5)
    {
        if(x==0)
            f=0;
        else
            f=pow((pow(x,5)+pow(fabs(x),0.5)),0.2);
        printf("%10.1f%10.4f\n", x, f);
        Dem++;
        x+=0.1;
        if(Dem%23==0) /* nếu hiện quá 23 dòng thì dừng lại*/
        {
            printf("Go Enter de xem tiep\n");
            getch();
        }
    }
    getch();
    return 0;
}
/* ******************************************************************************** */
```

Ví dụ 2-30. Viết chương trình mô phỏng *n* lần sinh và tính xem có mấy lần sinh con trai.

Giải: Việc dự đoán sinh con trai hay con gái được tính một cách ngẫu nhiên, nếu số phát ra nhỏ hơn *n/2* chẳng hạn (*n là một số cho trước*) thì được coi là sinh con trai, ngược lại là sinh con gái. Chương trình sử dụng hàm phát số ngẫu nhiên ***int random(int n)*** cho một số ngẫu nhiên từ *0* đến *n-1* và hàm *void randomize(void)* để khởi đầu bộ số ngẫu nhiên bằng một giá trị ngẫu nhiên. Các hàm này được khai báo trong ***stdlib.h***.

```
/* ******************************************************************************** */
#include "stdio.h"
#include "conio.h"
#include "stdlib.h"
/* ******************************************************************************** */
int main()
{
    int i,n, DemTrai;
    clrscr();
    printf("\nCan mo phong bao nhieu lan sinh ? \n");
    scanf("%d", &n);
    DemTrai=0;
    randomize();
    for(i=1; i<=n; i++)
    {
        if(random(n)<(int)(n/2))
```

```
            DemTrai++;
    }
    printf("\nSo lan sinh con trai trong %d lan sinh la: %d", n, DemTrai);
    getch();
    return 0;
}
/* ************************************************************************************* */
```

2.3. CÂU HỎI VÀ BÀI TẬP

1. Hãy so sánh biểu thức trong ngôn ngữ C với biểu thức trong toán học.

2. Biểu thức quan hệ và logic trong ngôn ngữ lập trình C và ngôn ngữ lập trình *Pascal* khác nhau ở điểm nào?

3. Việc chuyển đổi kiểu giá trị xảy ra khi nào? nguyên tắc hoạt động ra sao? Cho ví dụ minh hoạ.

4. Phân biệt các phép thao tác Bit *&*, */* với các phép logic *&&* và *//*. Cho ví dụ minh hoạ.

5. Phân biệt các câu lệnh *++i*, *--i* với *i++*, *i--*. Cho ví dụ minh hoạ.

6. Phân biệt vòng lặp xác định và vòng lặp không xác định? Cách sử dụng từng loại. Phân biệt toán tử ***switch*** với toán tử ***if*** mở rộng cho *n* lựa chọn, có thể viết lại ví dụ 2-16 sang cấu trúc dùng lệnh ***switch*** hay không? tại sao?

7 [*]. Ta có thể chuyển đổi cấu trúc ***rẽ nhánh if, switch*** và cấu trúc lặp ***for, do... while*** thành cấu trúc ***while*** được không? tại sao?

8. Hãy trình bày vai trò của các toán tử ***break, continue***.

9*. Tại sao khi viết chương trình bằng *Macro*, kích thước của chương trình lại lớn hơn nhưng nhìn chung chạy nhanh hơn là viết chương trình bằng hàm?

10*. Hãy viết lại ví dụ 2-11 mà không sử dụng hàm ***pow()***.

Gợi ý: ta có thể tính $f=y^x$ như sau:

- Trường hợp y>0 ta có: *f=exp(x*log(y))*
- Trường hợp y<0 ta có thể đưa về trường hợp y>0, đương nhiên cách viết còn phụ thuộc vào giá trị của x, chẳng hạn khi *x=1/(2n+1)* thì: *f=-exp(x*log(abs(y)))*.

Trong đó ***double exp(double x)*** sẽ cho giá trị e^x và ***double log(double x)*** sẽ cho giá trị *ln(x)*. Cả hai hàm này đều đã được khai báo trong thư viện ***math.h*** của Turbo C.

11. Hãy viết lại ***ví dụ 2-12*** có kiểm tra điều kiện hình thành tam giác.

12. Lập chương trình giải phương trình bậc hai. Đưa kết quả ra màn hình.

13. Nhập ba số a, b, c tính và in kết quả biểu thức sau đây ra màn hình (có biện luận các trường hợp vô nghĩa):

$$\frac{ab+4a-6}{3b-sin(bc)}+\frac{(1-a)(tg(b-2)+3)}{\cos a+\sin b}$$

[*] Là các bài khó cần suy nghĩ nhiều.

14. Nhập tọa độ hai điểm từ bàn phím. Tính và in khoảng cách giữa hai điểm đó ra màn hình.

15. Nhập tọa độ một điểm từ bàn phím. Xét xem điểm đó có thuộc đường phân giác của góc phần tư thứ nhất hay không?

16. Nhập ba biến số nguyên từ bàn phím. Hãy xét xem ba số đó có tạo thành cấp số nhân hay không?

17. Xác định số lần lặp của chu trình trong chương trình sau đây:

```
#include "stdio.h"
#include "conio.h"
int main()
{
        int a=1, b;
        float x=0;
        b=a;
        while(a||b)
        {
              x=2*x+1;
              if(x>0)
              {
                    a=0;
                    if(x<200)
                          b=0;
              }
              printf("\nx=%f",x);
        }
        getch();
}
```

18. Nhập *n* số từ bàn phím. Tính và in ra màn hình giá trị của số lớn nhất trong *n* số đó.

19. Nhập ba số *a, b, c* từ bàn phím. Hãy tính và đưa kết quả ra màn hình giá trị lớn nhất trong các biểu thức sau: *1-2sina*, cos^2b *và ab-sinacosb*.

20. Nhập giá trị của *x* từ bàn phím. Tính và in giá trị biểu thức sau đây ra màn hình:

$$f(x)=\begin{cases} 2x+3 & \text{nếu } x < -5 \\ \cos(2x+1)+x & \text{nếu } -5 <= x < 10 \\ 4x+7 & \text{nếu } x >= 10 \end{cases}$$

21. Cho bốn số nguyên *a, b, c, d*. Đặt S1= *a+b+c+d*, S2= *ab-cd*. Hãy kiểm tra xem S1 có chia hết cho S2 hay không? Nếu chia hết thì in ra thương số, ngược lại in ra số dư.

22. Lập chương trình giải bất phương trình $ax^2+bx+c<0$ với *a*, *b*, *c* là các hệ số nhập từ bàn phím.

23*. Tính và in tổng theo công thức sau:

$$S = 2004 + 1 - \frac{1}{2!} + \frac{1}{3!} - \frac{1}{4!} ... + (1)^{(n-1)} \frac{1}{n!}$$ với *n* là một số nguyên dương bất kì.

24*. Tính và in tổng theo công thức sau:

$$S = 1 + x + \frac{x^2}{2} + \frac{x^3}{3} + ... + \frac{x^n}{n}$$ với x là một số thực nhập từ bàn phím.

25*. Lập chương trình tính giá trị của e^x theo công thức xấp xỉ sau:

$$e^x = 1 + \frac{x}{1!} + \frac{x^2}{2!} + ... + \frac{x^n}{n!}$$ với độ chính xác ε nhập từ bàn phím.

26. Lập chương trình đọc vào hai số *a*, *b* rồi tính $y=15x^2+x+72$, trong đó:

x=a+b nếu a<b

x=15,172 nếu a=b

x=a-b nếu a>b

27. Lập chương trình đọc *x* từ bàn phím rồi tính: $Y = \sqrt{|z-1|} + \frac{1}{z^2+1}$

$$\text{trong đó } z = \begin{cases} 2\sin^2(6{,}14 + x) & \text{nếu } x > 0 \\ \dfrac{\sqrt{x}\sin x}{e^{-x} + x} & \text{nếu } x <= 0 \end{cases}$$

28*. Nhập vào một số thực từ 0,01 đến 99,99. Hãy in ra màn hình cách đọc số đó theo tiếng Việt. Ví dụ 12,22 thì in ra "Muoi hai phay hai muoi hai".

29. Viết chương trình đảo các số trong một số nguyên cho trước. Ví dụ nhập vào số 12345 thì phải in ra 54321.

30. Viết chương trình xếp các dấu * thành một hình thoi *n* dòng, với n đọc từ bàn phím.

31*. Viết chương trình phân tích số nguyên *n* (*n nhập từ bàn phím*) thành thừa số nguyên tố.

32. Lập chương trình cho phép nhập từ bàn phím các kí tự, chương trình sẽ thoát khi bấm phím *ESC*. Hãy đếm xem đã gõ bao nhiêu phím là chữ hoa, bao nhiêu phím là chữ thường. Đưa kết quả ra màn hình khi kết thúc nhập.

33. Lập chương trình mô phỏng việc gieo cùng một lúc 2 xúc xắc *n* lần và đếm số lần xuất hiện tổng số điểm bằng *6*.

34*. Viết lại ví dụ 2-15 để có thể trở thành một chương trình tư vấn tình cảm cho mọi người. Nội dung tư vấn tùy thích.

Chương 3

TỔ CHỨC CHƯƠNG TRÌNH VỀ MẶT DỮ LIỆU

MỤC TIÊU CỦA CHƯƠNG NÀY

- *Hiểu mối tương quan giữa cấu trúc dữ liệu và giải thuật.*
- *Hiểu cách hoạt động cũng như cách lưu trữ của các loại cấu trúc dữ liệu thường gặp trong bộ nhớ.*
- *Sử dụng thành thạo các cấu trúc dữ liệu cũng như giải thuật cơ bản.*

3.1. CẤU TRÚC DỮ LIỆU, GIẢI THUẬT VÀ CÁC VẤN ĐỀ LIÊN QUAN

3.1.1. Thuật toán và giải thuật

1. Thuật toán

Khái niệm thuật toán (*algorithm*) bắt nguồn từ tên một nhà toán học người Trung Á là *Abu Abd - Allah ibn Musa al'Khwarizmi*, thường gọi là *Al'Khwarizmi*. Trong một cuốn sách viết về số học Ông đã dùng phương pháp mô tả rất rõ ràng, mạch lạc cách giải của một số bài toán. Về sau, phương pháp mô tả cách giải toán của Ông được xem là chuẩn mực và được nhiều nhà toán học tuân theo. Khái niệm *Algorithm* ra đời dựa theo cách phiên âm tên của Ông.

Định nghĩa:

Thuật toán là một dãy các bước chặt chẽ và rõ ràng, xác định một trình tự các thao tác trên một số đối tượng nào đó sao cho sau một số hữu hạn lần thực hiện ta thu được kết quả như mong đợi.

Việc nghiên cứu về thuật toán có một vai trò rất lớn trong khoa học máy tính, mọi vấn đề, bài toán muốn được thực hiện trên máy tính điện tử đều phải có một *thuật toán* hoặc *giải thuật* xác định cho nó, đồng thời kết quả thực hiện này cũng phụ thuộc rất nhiều vào *thuật toán* hay *giải thuật* đã được sử dụng. Trong khoa học máy tính, mỗi *thuật toán* thường được thể hiện bởi một thủ tục gồm một số hữu hạn các câu lệnh mà theo đó ta sẽ đạt đến lời giải cho bài toán đang xét.

2. Giải thuật

Trong khi tìm kiếm lời giải cho các bài toán thực tế, các nhà khoa học nhận thấy rằng:

- Có nhiều bài toán cho đến nay vẫn chưa xác định được liệu có tồn tại một thuật toán để giải quyết hay không ?

- Có nhiều bài toán đã có thuật toán để giải, nhưng không chấp nhận được do thời gian để giải bài toán theo thuật toán đó quá lớn hoặc các điều kiện cho thuật toán khó đáp ứng.

- Có những bài toán có thể giải được một cách hữu hiệu bằng một lời giải nào đó, nhưng lời giải này lại vi phạm một số tính chất của thuật toán.

Trong thực tiễn, có rất nhiều trường hợp người ta chấp nhận các cách giải thường cho kết quả tốt (*tất nhiên là không phải lúc nào cũng tốt*) nhưng ít phức tạp, hiệu quả và khả thi. Để có thể dự báo thời tiết một cách chính xác cho ngày hôm sau, thông thường người ta phải giải một bài toán tối ưu với khoảng *15* ẩn số, theo tính toán của các nhà khoa học thì thời gian để thực hiện công việc này theo đúng *thuật toán* phải mất nhiều năm, như vậy kết quả thu được sẽ không còn có giá trị gì nữa. Trong thực tế, người ta có những cách giải khác đơn giản hơn rất nhiều và cho kết quả cũng tương đối khả quan, đương nhiên kết quả đôi khi có thể không đúng vì nó vi phạm tính chặt chẽ của thuật toán. Do đó người ta đã nghĩ đến việc mở rộng khái niệm thuật toán, làm cho thuật toán bớt "*cứng nhắc*" hơn, không đòi hỏi quá chặt chẽ và rõ ràng mà vẫn cho kết quả ***chấp nhận được***. Chấp nhận được ở đây có thể hiểu như một kết quả gần đúng, một kết quả gần sát với thực tế nhưng khả thi, hoặc một kết quả đã bị ràng buộc trong một số điều kiện nhất định nào đó... Những ***cách giải chấp nhận được*** nhưng không hoàn toàn đáp ứng đầy đủ các tính chất của một thuật toán như thế gọi là ***giải thuật***.

Giải thuật đệ quy (*xem chương 4*), giải thuật *ngẫu nhiên*, các giải thuật *Heuristic*... chính là sự mở rộng của khái niệm *thuật toán*. Để thuận tiện trong việc sử dụng ngôn từ, trong giáo trình này chúng tôi sẽ dùng khái niệm *giải thuật* để chỉ chung cho *thuật toán* và *giải thuật*.

3.1.2. Cấu trúc dữ liệu và các vấn đề liên quan

Cấu trúc dữ liệu và giải thuật là hai thành tố có mối quan hệ gắn bó chặt chẽ với nhau trong một chương trình máy tính. Việc tìm kiếm một cấu trúc dữ liệu và cùng với nó là các cấu trúc điều khiển để viết một chương trình giải một bài toán nào đó phụ thuộc rất nhiều vào *giải thuật* để giải bài toán đó. Ngược lại mỗi giải thuật đưa ra cũng phải quan tâm đến việc nó sẽ xử lí trên các đối tượng dữ liệu nào? (*trong một số ví dụ ứng dụng sẽ phân tích kĩ hơn mối quan hệ này*). Ta có thể trích dẫn một câu nói nổi tiếng của *N. Wirth* như sau:

Chương trình = Giải thuật + Cấu trúc dữ liệu
(Program = Algorithm + Data-structure)

Thông thường, mỗi một ngôn ngữ lập trình bao giờ cũng cung cấp sẵn các cấu trúc dữ liệu tiền định (*kiểu số nguyên, số thực, kí tự...*) cho người lập trình, tuy nhiên các cấu trúc này là khác nhau với các ngôn ngữ lập trình khác nhau. Nhưng trong hầu hết các trường hợp ta đều phải tự xây dựng lên các cấu trúc dữ liệu riêng dựa trên các kiểu dữ liệu đã có để có thể đáp ứng được các yêu cầu đặt ra. Mỗi cấu trúc dữ liệu này đều có ***nguyên tắc hoạt động***, các ***phép tác động*** và ***cách thức lưu trữ*** riêng. Do đó khi nghiên cứu bất kì một cấu trúc dữ liệu nào ta đều phải nghiên cứu trên cả ba phương diện đó.

3.2. CÁC CẤU TRÚC DỮ LIỆU CƠ BẢN

3.2.1. Con trỏ

1. Con trỏ, kiểu con trỏ, kiểu địa chỉ

Con trỏ là một biến đặc biệt dùng để chứa địa chỉ của *biến* và *hàm*. Vì có rất nhiều loại *biến* và *hàm* khác nhau cho nên cũng có nhiều loại con trỏ khác nhau. Ví

dụ *con trỏ kiểu **int*** dùng để chứa *địa chỉ các biến kiểu **int***, *con trỏ kiểu **float*** dùng để chứa *địa chỉ các biến kiểu **float***... Ta nói rằng, mỗi *kiểu con trỏ* sẽ được gán tương ứng với một *kiểu địa chỉ* nhất định. Cách làm việc của các con trỏ ứng với các kiểu địa chỉ cũng rất khác nhau (*xem ví dụ 3-2*). Việc gán các con trỏ với các kiểu địa chỉ khác kiểu sẽ dẫn tới việc nhận được các cảnh báo không mong muốn và có thể gây ra kết quả sai, lỗi biên dịch, thậm chí treo máy. Do đó khi sử dụng con trỏ ta cần hết sức thận trọng, mặc dầu vậy, việc sử dụng con trỏ là một trong những sức mạnh tiềm tàng của ngôn ngữ lập trình *C* cần được khai thác triệt để.

2. Cách sử dụng con trỏ

Cũng như các biến khác, trước khi sử dụng ta phải khai báo con trỏ. Con trỏ trong ngôn ngữ lập trình C được khai báo theo cú pháp sau đây:

```
Kiểu *TenBienTro;
```

Ví dụ để khai báo một con trỏ kiểu số thực ta có thể viết như sau:

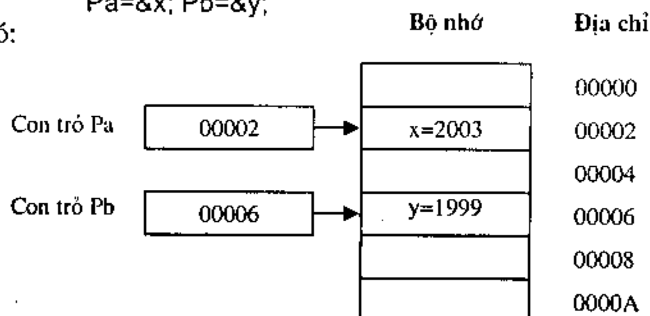
```
float *P; /* P là tên của con trỏ */
```

Mối quan hệ giữa con trỏ và biến có thể được mô tả như sau:

Ví dụ 3-1. Với khai báo :

```
int *Pa, Pb, x, y;
x=2003; y=1999;
Pa=&x; Pb=&y;
```

Thì ta có:



Hình 3.1. Mối quan hệ giữa con trỏ và biến.

Khi con trỏ ***Pa*** chứa địa chỉ của biến ***x*** ta nói rằng ***Pa*** trỏ đến ***x*** và ta có thể truy xuất đến biến ***x*** thông qua con trỏ này. Tuy nhiên để truy xuất giá trị của một biến thông qua một con trỏ ta phải dùng ***dạng khai báo*** của chính con trỏ đó. Ví dụ, để gán giá trị *123* cho biến ***x*** ở trên ta có thể viết như sau:

```
*Pa=123; /* *Pa chính là dạng khai báo của con trỏ Pa */
```

Khi đó giá trị của bản thân biến ***x*** bị thay đổi tương đương với việc dùng câu lệnh: `x=123;`

Ví dụ 3-2. Ví dụ về một số kiểu con trỏ và kiểu địa chỉ.

Với các khai báo sau :

```
float *Ten1[100];
float (*Ten2)[100];
float  *Ten3;
```

float Ten4[25][100];

Câu lệnh thứ nhất sẽ khai báo ra một *mảng con trỏ* (*phần sau ta sẽ nghiên cứu kĩ về mảng*) kiểu số thực có tên là ***Ten1*** gồm *100* phần tử. Câu lệnh thứ hai sẽ khai báo ra một *con trỏ* kiểu ***float[100]*** có tên là ***Ten2*** (nghĩa là nó sẽ chứa được kiểu địa chỉ *float[100]*). Câu lệnh thứ ba đơn thuần chỉ khai báo một biến trỏ kiểu *float* có tên là ***Ten3***. Còn câu lệnh thứ tư dùng để khai báo một mảng hai chiều gồm *25* hàng, mỗi hàng có *100* phần tử kiểu ***float*** có tên là ***Ten4*** và ***Ten4*** được coi là một ***hằng địa chỉ kiểu float[100]*** (*phần sau sẽ giải thích kĩ hơn vấn đề này*). Như vậy nếu ta thực hiện phép gán:

Ten3=Ten4; sẽ bị cảnh báo vì ta gán sai kiểu địa chỉ cho con trỏ ***Ten3***.

Ten1=Ten4; sẽ báo lỗi vì ***Ten1*** là một hằng địa chỉ kiểu con trỏ ***float*** còn ***Ten4*** cũng là một hằng địa chỉ kiểu ***float[100]***.

Nhưng câu lệnh

Ten2=Ten4; thì hoàn toàn hợp lệ vì ta đã gán một hằng địa chỉ kiểu ***float[100]*** cho một con trỏ kiểu ***float[100]***.

Các phép toán trên con trỏ

a) Phép cộng, trừ địa chỉ

Trong ngôn ngữ lập trình C, ta có thể cộng, trừ giá trị của một con trỏ (*chính là địa chỉ của một vùng nhớ nào đó*) với một số nguyên để cho kết quả cũng là một *giá trị địa chỉ cùng kiểu*. Để hiểu rõ điều đó ta hãy xét ví dụ dưới đây.

Ví dụ 3-3. Xét đoạn chương trình:

```
int *Pix, *Piy, x;
float *Pfx,*Pfy, y;
Pix= &x; /* cho Pix trỏ đến biến x (chứa địa chỉ vùng nhớ của biến x)*/
Piy= Px+1; /* Piy sẽ trỏ đến biến nguyên nằm sau x trong bộ nhớ, có nghĩa
là giá trị của nó được tăng lên 2 (bằng kích thước của biến nguyên) */
x= Piy - Pix; /* sẽ gán 1 cho x, tuy nhiên phép cộng, nhân, chia... thì không
hợp lệ trong trường hợp này */
Pfy= &y; /* Pfy sẽ trỏ đến biến thực y */
Pfx= Pfy-1; /* Pfx sẽ trỏ đến biến thực nằm ngay trước biến y trong bộ nhớ,
có nghĩa là giá trị của nó bị giảm đi 4 (bằng kích thước của biến thực) */
```

Điều đó có nghĩa là phép cộng (*trừ*) địa chỉ đối với mỗi *kiểu con trỏ* sẽ thay đổi tuỳ theo *kiểu địa chỉ* mà nó chứa (*kể cả kiểu dữ liệu do người dùng định nghĩa*).

b) Phép gán

Ta có thể gán cho một con trỏ địa chỉ của một biến cùng loại (*như các ví dụ trên*) hoặc giá trị của một con trỏ cùng loại khác. Ví dụ, để con trỏ ***Piy*** cũng trỏ đến biến ***x*** ở trên ta có thể viết: ***Piy=Pix***; Để có thể gán giá trị của các con trỏ khác loại cho nhau ta phải dùng toán tử ép kiểu *()*. Chương trình sau đây minh họa cách dùng của toán tử ép kiểu để truy nhập *Byte cao* và *Byte thấp* của một biến nguyên:

Ví dụ 3-4. Viết chương trình gán giá trị cho từng byte của một biến nguyên, đưa ra màn hình giá trị của số nguyên đó.

```
/* ********************************************************************************** */
#include "stdio.h"
```

```
#include "conio.h"
/* ******************************************************************************** */
int main()
{
    int BienNguyen;
    char *Pc;
    Pc=(char*) &BienNguyen; /* Báo cho trình biên dịch biết là ta làm việc với
    vùng nhớ của biến nguyên theo từng byte */
    *Pc= 'a' ; /* đưa giá trị 61h vào Byte thấp của biến nguyên BienNguyen */
    *(Pc+1) = 'b' /* đưa giá trị 62h vào Byte cao của biến nguyên BienNguyen */
    printf("Bien nguyen co gia tri la %X H", BienNguyen);
    getch();
    return 0;
}
/* ******************************************************************************** */
```

Kết quả chạy chương trình:

```
    Bien nguyen co gia tri la 6261 H
/* ******************************************************************************** */
```

c) Phép so sánh con trỏ

Đối với con trỏ ta có thể thực hiện các phép so sánh ==, <=... với các con trỏ khác (*cùng kiểu*) hoặc với giá trị NULL (*là giá trị đặc biệt để chỉ ra rằng con trỏ chưa trỏ đến một vùng nhớ cụ thể nào*).

Ví dụ 3-5. Minh hoạ các phép so sánh con trỏ. Xét đoạn chương trình sau:

```
...
int *Pa, *Pb;
...
if((Pa==NULL)&&(Pb==NULL))
        printf("\n Pa va Pb chua tro den vung nho nao !");
else if(Pa<Pb)
        printf("\n Vùng nho cua Pa thap hon vung nho cua Pb !");
else if(Pa>Pb)
        printf("\n Vùng nho cua Pa cao hon vung nho cua Pb !");
else    printf("\n Pa va Pb tro den cung mot vung nho");
...
```

Chú ý:

- Khi một biến trỏ đã được khai báo nhưng chưa được gán địa chỉ của một vùng nhớ nào đó thì vẫn chưa sử dụng được (*nếu cố tình sử dụng có thể gây ra những lỗi không biết trước*). Ví dụ sau đây có thể làm rõ nguyên tắc này:

```
char *Name, HoTen[25];
gets(Name);
```

Câu lệnh này đúng về mặt cú pháp nhưng sẽ phát sinh một cảnh báo khi chương trình thực hiện. Mục đích của câu lệnh này là đọc từ bàn phím một chuỗi kí tự và lưu vào vùng nhớ do con trỏ ***Name*** trỏ tới. Thế nhưng con trỏ ***Name*** vẫn chưa trỏ đến một vùng nhớ nào cả. Ta có thể sửa lại như sau:

```
char *Name, HoTen[25];
```

```
Name=HoTen;
gets(Name);
```

- Khi một biến trỏ chưa trỏ đến một vùng nhớ nào ta nên gán cho nó một giá trị *NULL* để tránh truy nhập vào những vùng nhớ không biết trước do quên gán giá trị cho nó.

3. Con trỏ không kiểu (con trỏ void)

Con trỏ kiểu ***void*** là con trỏ đặc biệt có thể nhận bất kì kiểu địa chỉ nào. Ví dụ như sau:

```
void *Px,*Py;
int x=19;
float y= 65;
Px=&x;
Py=&y;
```

Các phép toán cộng, trừ, so sánh và sử dụng dạng khai báo không áp dụng được với con trỏ *void*. Trong ví dụ *4-13* của chương 4 ta sẽ xem xét cụ thể cách làm việc của loại con trỏ này và ứng dụng của nó trong thực tiễn.

3.2.2. Mảng

1. Khái niệm và đặc trưng

Mảng là một tập hợp ***có thứ tự*** gồm một số ***cố định*** các phần tử của ***cùng một kiểu dữ*** liệu nào đó. Kiểu dữ liệu này có thể là các kiểu dữ liệu nguyên thuỷ trong ngôn ngữ lập trình (*kiểu integer, real... trong ngôn ngữ Pascal*; *kiểu int, long, float... trong ngôn ngữ lập trình C...*) hoặc là các kiểu dữ liệu do người sử dụng tự định nghĩa ra.

Các đặc trưng cơ bản của mảng:

- Độ dài mảng (*kích thước của mảng và thường kí hiệu là **n***) là cố định. Điều đó có nghĩa là khi một mảng đã được khai báo, ta không thể thao tác *bổ sung* hoặc *loại bỏ* các phần tử của mảng. Do đó, khi lưu trữ các đối tượng mà có số lượng luôn biến động thì cấu trúc dữ liệu kiểu mảng thường ít khi được sử dụng.

- Các phần tử của một mảng phải cùng kiểu. Điều đó có nghĩa là mảng chỉ có thể lưu trữ giá trị cho các loại đối tượng cùng loại mà thôi, nếu các đối tượng được lưu trữ khác nhau, thì ta không thể chọn mảng làm cấu trúc dữ liệu mà phải chọn các loại cấu trúc dữ liệu khác.

- Mỗi phần tử của mảng bao giờ cũng được đặc trưng bởi bốn tham số là: ***tên mảng***, ***chỉ số***, ***kiểu*** và ***giá trị***. Trong đó, *giá trị* là biểu diễn của đối tượng được lưu trữ trong mỗi phần tử của mảng, *chỉ số* là *thứ tự* của phần tử đó trong mảng.

2. Cấu trúc lưu trữ và cách sử dụng mảng một chiều (véc tơ)

a. Cách khai báo

```
Kiểu  TênMảng[KíchThước];
```

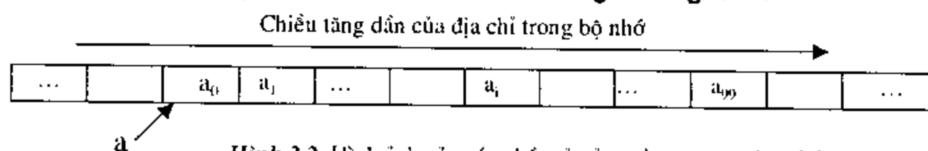
Trong đó, *KíchThước* sẽ chỉ ra số phần tử tối đa trong mảng.

Ví dụ để khai báo một mảng các số thực có tên ***Mang100*** gồm *100* phần tử ta có thể viết như sau: float Mang100[100];

b. Cấu trúc lưu trữ của véc tơ trong bộ nhớ

Với khai báo *float a[100];* thì trình biên dịch sẽ cấp cho chương trình ứng dụng một vùng nhớ có kích thước là *4 x 100* bytes *liên tiếp nhau trong bộ nhớ* bắt đầu từ một địa chỉ xác định nào đó, địa chỉ này sẽ được gán cho tên của mảng là *a* (*có nghĩa rằng, tên mảng là một hằng con trỏ chứa địa chỉ của phần tử đầu tiên trong mảng*). Do đó muốn truy nhập đến các phần tử khác của mảng ta có thể truy nhập thông qua địa chỉ nằm trong tên mảng theo công thức sau:

Địa chỉ của phần tử có chỉ số *i* của mảng *a* bằng *a + i* [21]



Hình 3.2. Hình ảnh của các phần tử của mảng a trong bộ nhớ.

c. Cách truy nhập các phần tử của véc tơ

Để có thể truy xuất đến từng phần tử của mảng, ta có thể truy xuất theo hai cách. ***Cách thứ nhất***, truy xuất trực tiếp thông qua *tên mảng* và *chỉ số*. Ví dụ để gán giá trị *10.0* cho phần tử thứ *5* của mảng trên ta có thể viết như sau: a[4]=10;

Cách thứ hai, truy xuất thông qua việc tính địa chỉ của phần tử cần truy xuất trong bộ nhớ. Ví dụ để đưa giá trị *2003* vào phần tử thứ *9* của mảng trên đây (*có chỉ số i ==8*) ta có thể viết: *(a+8)=2003; trong đó (a+8) sẽ cho ta địa chỉ của phần tử thứ *9* của mảng. Còn phép *(a+8) sẽ cho phép ta truy xuất đến ô nhớ đặt tại địa chỉ (a+8) và do đó giá trị *2003* sẽ được gán cho phần tử thứ *9* của mảng.

Những điều trên đây có thể viết tổng quát lại như sau:

Với mảng ***a[100]*** thì các cách viết sau là tương đương:

```
a          hoặc    &a[0]
a+i        hoặc    &a[i]
*(a+i)     hoặc    a[i]
Với 0<=i<100
```

Đối với mảng ngoài hoặc mảng tĩnh (*cả trong và ngoài*) thì ta có thể khởi đầu (*gán giá trị ngay lúc khai báo*) *một lần vào lúc dịch chương trình* cho chúng. Trong khi khởi đầu cho mảng ngoài hoặc mảng tĩnh ta có thể không cần chỉ rõ kích thước của chúng. Khi đó máy sẽ dành cho mảng một khoảng nhớ đủ lớn để thu nhận danh sách các giá trị khởi đầu.

Ví dụ 3-6. Đoạn chương trình sau đây sẽ chỉ rõ cách thức khởi đầu cho mảng ngoài và mảng tĩnh.

```
...
float MangThuc100[100]= {2.1, 0, 4.5, 36};
```

[21] Nếu *a* là mảng nguyên thì *a* sẽ chứa kiểu địa chỉ nguyên (có nghĩa là *a+1* sẽ trỏ đến 2 bytes tiếp theo), nếu *a* là mảng thực thì *a* sẽ chứa kiểu địa chỉ thực (có nghĩa là *a+1* sẽ trỏ đến 4 bytes tiếp theo)...

```
int main()
{
        static int MangNguyen[]= { 3, 5, 6, 8};
        ...
}
...
```

Các câu lệnh trên sẽ khai báo ra một mảng các số thực có tên là ***MangThuc100*** gồm *100* phần tử với bốn phần tử đầu tiên được khởi gán các giá trị tương ứng là *2.1, 0, 4.5, 3.6* còn các phần tử khác có giá trị là *0* và một mảng tĩnh trong gồm bốn phần tử nguyên có tên là ***MangNguyen*** và được khởi gán các giá trị tương ứng là *3, 5, 6 và 8*.

Chú ý:

- Mảng thực chất là một loại biến trong các ngôn ngữ lập trình, do đó nó có mọi tính chất và đặc trưng của một biến.
- Phần tử đầu tiên của mảng trong ngôn ngữ lập trình *C* bắt đầu từ chỉ số *0*.
- Việc truy nhập mảng theo cách hai sẽ được ứng dụng nhiều trong việc truyền *tham số là mảng* cho hàm (*xem chương 4*).

Ví dụ 3-7. Nhập vào một dãy ***n*** số thực (***n** nhập từ bàn phím*), tính và đưa ra màn hình trung bình cộng của các phần tử có giá trị lớn hơn 4.

```
/* ***************************************************************************************** */
#include "stdio.h"
#include "conio.h"
#define MAX  50
/* ***************************************************************************************** */
int main()
{
    float DaySo[MAX], TBC= 0;
    int n, Dem=0,i;
    printf("\nHay cho biet can nhap bao nhieu so n=  ");
    scanf("%d",&n);
    for(i=0;i<n;++i)
    {
        printf("\nHay nhap gia tri cho phan tu:  DaySo[%d]= ", i);
        scanf("%f",&DaySo[i]);
        if(DaySo[i]>4)
        {
            TBC+=DaySo[i];
            Dem++;
        }
    }
    printf("\nTrung binh cong cua cac so co gia tri > 4 trong day la:%f", TBC/Dem);
    getch();
    return 0;
}
/* ***************************************************************************************** */
```

Bài tập:

- Hãy giải thích hoạt động của chương trình trên.
- Hãy viết lại chương trình trên theo cách truy xuất thứ 2.

Ví dụ 3-8. Viết chương trình nhập vào từ bàn phím *12* số có giá trị nhỏ hơn *80* đặc trưng cho năng suất của *12* tháng rồi dùng kí tự '*' để vẽ biểu đồ ngang lên màn hình.

```
/* ********************************************************************************** */
#include "stdio.h"
#include "conio.h"
/* ********************************************************************************** */
int main()
{
    int NangSuat[12], i, j;
    clrscr();
    printf("\nHay nhap nang suat cua 12 thang:");
    for(i=0; i<12; ++i)
    {
        printf("\nThang thu %d = ", i);
        do
        {
            /* Chỉ nhập giá trị nhỏ hơn 80*/
            scanf("%d", &NangSuat[i]);
        }
        while(NangSuat[i]>=80);
    }
    clrscr();
    printf("\nBieu do nang suat 12 thang\n");
    for(i=0; i<12; ++i)
    {
        for(j=0; j<NangSuat[i]; ++j)
            printf("%c", '*');
        printf("\n");
    }
    getch();
    return 0;
}
/* ********************************************************************************** */
```

3. Cấu trúc lưu trữ và cách sử dụng của mảng hai chiều (*ma trận*)

a. Cách khai báo

Cũng tương tự như với mảng một chiều, trước khi sử dụng, mảng hai chiều cũng phải được khai báo theo cú pháp sau đây:

Kiểu Tên_Mảng[Số_Dòng][Số_Cột];

Ví dụ câu lệnh float MaTranThuc[20][30]; sẽ khai báo một ma trận các số thực có tên là ***MaTranThuc*** gồm có 20 hàng và 30 cột (*600 phần tử*).

b. Cấu trúc lưu trữ của ma trận trong bộ nhớ

Đối với *ma trận*, người ta cũng sử dụng cách thức lưu trữ kế tiếp như đối với mảng một chiều, nhưng tuỳ từng ngôn ngữ lập trình khác nhau mà việc lưu trữ sẽ

được tiến hành theo *thứ tự ưu tiên hàng* hay *thứ tự ưu tiên cột*. Trong ngôn ngữ lập trình C, ma trận được lưu trữ theo thứ *tự ưu tiên hàng*.

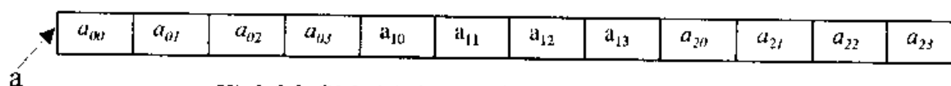
Lưu trữ theo *thứ tự ưu tiên hàng* có nghĩa là ma trận sẽ được lưu trữ kế tiếp nhau trong bộ nhớ, *hết hàng này đến hàng khác*, bắt đầu từ một địa chỉ xác định nào đó, địa chỉ này sẽ được chứa trong ***tên của biến ma trận***.

Ví dụ 3-9. Minh họa cách lưu trữ của ma trận cấp *3x4*:

Với khai báo float a[3][4];

Thì ta nhận được:

$$a_{i,j} = \begin{pmatrix} a_{00} & a_{01} & a_{02} & a_{03} \\ a_{10} & a_{11} & a_{12} & a_{13} \\ a_{20} & a_{21} & a_{22} & a_{23} \end{pmatrix}$$



Hình 3.3. Cách thức lưu trữ của ma trận a trong bộ nhớ

Do trong ngôn ngữ lập trình C, ma trận được lưu trữ theo thứ tự ưu tiên hàng, cho nên về thực chất ma trận trên được *coi như* là một véc tơ gồm *3* phần tử (*chính là số hàng của ma trận*), mỗi phần tử lại là một véc tơ gồm *4* số thực liên tiếp nhau (*là số cột của ma trận, nó cũng chính là số phần tử trên mỗi hàng*).

Vì thế khi ta viết ***a*** sẽ cho địa chỉ của phần tử đầu tiên thuộc hàng thứ nhất (*phần tử **a[0][0]***) của ma trận ***a***. Nhưng ***a+1*** lại cho địa chỉ của phần tử đầu tiên thuộc hàng thứ 2 (*phần tử **a[1][0]***) của ma trận. Có nghĩa là tên ma trận ***a*** trong trường hợp này sẽ là một *hằng con trỏ* chứa ***kiểu địa chỉ float[4]***.

Chính vì vậy, muốn tính được địa chỉ bên trong bộ nhớ của phần tử ở hàng có chỉ số ***i*** và cột có chỉ số ***j*** của ma trận trên (*chính là phần tử a[i][j]*) phải *báo* cho trình biên dịch Turbo C biết là ta đang muốn làm việc với ***kiểu địa chỉ float*** (*chứ không phải kiểu địa chỉ float[4]*) bằng phép *ép kiểu* như sau:

(float *) a + i *4 + j

c. Cách truy nhập đến từng phần tử của ma trận

Tương tự như véc tơ, ta cũng có thể truy xuất các phần tử của ma trận bằng hai cách. *Cách thứ nhất* đó là *truy xuất trực tiếp* thông qua *tên ma trận* và *chỉ số dòng, chỉ số cột* theo cú pháp như sau: *a[2][3] = 2003. ;* với câu lệnh này giá trị *2003* sẽ được chuyển vào phần tử ở hàng thứ *3* và cột thứ *4* của ma trận. *Cách thứ hai* là truy xuất thông qua *địa chỉ* của từng phần tử trong ma trận. Tuy nhiên, ngôn ngữ lập trình C ***không cho phép lấy địa chỉ trực tiếp*** của một phần tử của ma trận thông qua toán tử lấy địa chỉ *&* (*ngoại trừ ma trận nguyên*). Ví dụ chương trình dưới đây nhằm vào số liệu cho một ma trận số thực khi chạy sẽ cho một thông báo lỗi như sau :

```
scanf : floating point formats not linked
Abnormal program termination
```

Và chúng ta không thể thực hiện tiếp chương trình như dự kiến.

Ví dụ 3-10. Ví dụ sai về việc truy nhập các phần tử của ma trận.

```
/* ************************************************************************************ */
#include"stdio.h"
int main()
{
	float a[10][5];
	int i, j;
	for(i=0; i<10; ++i) /* xét tất cả các dòng từ dòng 0 đến dòng 9 */
		for(j=0; j<5; ++j) /* vào số liệu cho từng phần tử của mỗi dòng */
		{
			printf("\nHay nhap du lieu cho a[%d][%d]= ", i, j);
			scanf("%f",&a[i][j]);
		}
	return 0;
}
/* ************************************************************************************ */
```

Để giải quyết được vấn đề này, ta có hai giải pháp. *Giải pháp thứ nhất* là dùng biến trung gian. Theo cách này ta có thể viết lại đoạn chương trình như sau:

Ví dụ 3-11. Sử dụng biến trung gian để nhập dữ liệu cho ma trận.

```
/* ************************************************************************************ */
#include"stdio.h"
#include"conio.h"
/* ************************************************************************************ */
int main()
{
	float a[10][5], tg;
	int i, j;
	for(i=0; i<10; ++i) /* xét tất cả các dòng từ dòng 0 đến dòng 9 */
		for(j=0; j<5; ++j) /* vào số liệu cho từng phần tử của mỗi dòng */
		{
			printf("\nHay nhap du lieu cho a[%d][%d]= ", i, j);
			scanf("%f",&tg); /* nhập dữ liệu vào biến trung gian */
			a[i][j] = tg;
		}
	for(i=0; i<10; ++i) /* xét tất cả các dòng từ dòng 0 đến dòng 9 */
		for(j=0; j<5; ++j) /* hiển thị từng phần tử của mỗi dòng */
			printf("a[%d][%d]=  %10.2f", i, j, a[i][j]);
	getch();
	return 0;
}
/* ************************************************************************************ */
```

Giải pháp thứ hai là truy xuất thông qua việc tính địa chỉ cụ thể của từng phần tử trong ma trận như đã trình bày ở công thức trên và đoạn chương trình có thể được viết lại như sau:

Ví dụ 3-12. Nhập dữ liệu cho ma trận thông qua việc tính địa chỉ.

```
/* ************************************************************************************ */
#include"stdio.h"
#include"conio.h"
```

```
/* ****************************************************************************** */
int main()
{
      float a[10][5], tg;
      int i, j;
      for(i=0; i<10; ++i) /* xét tất cả các dòng từ dòng 0 đến dòng 9 */
            for(j=0; j<5; ++j) /* vào số liệu cho từng phần tử của mỗi dòng */
            {
                  printf("\nHay nhap du lieu cho a[%d][%d]= ", i, j);
                  scanf("%f", (float*) a + i * 5 +j );
            }
      for(i=0; i<10; ++i) /* xét tất cả các dòng từ dòng 0 đến dòng 9 */
            for(j=0; j<5; ++j) /* hiển thị từng phần tử của mỗi dòng */
                  printf("a[%d][%d]=  %10.2f", i, j, a[i][j]);
      getch();
      return 0;
}
/* ****************************************************************************** */
```

Đối với ma trận ta cũng có thể khởi đầu theo cú pháp sau đây:

```
int MaTranNguyen[][6]= {
                              {1,2,3},
                              {4,5},
                              {6,7,8,9}
                        };
float MaTranThuc[10][10]= {
                              {10,11},
                              {1,3,6,8},
                              {0}
                        };
```

Chú ý:

- Khi chỉ ra kích thước mảng, thì kích thước này phải lớn hơn kích thước của bộ khởi đầu.

- Khi có một kích thước vắng mặt, thì kích thước vắng mặt phải nằm bên tay trái nhất (*ví dụ như khởi đầu cho MaTranNguyen ở trên*).

- Bộ khởi đầu của một mảng kiểu ***char*** có thể:

+ *Hoặc là danh sách các kí tự*

+ *Hoặc là một hằng xâu kí tự.*

Ví dụ: char Name[]={'M','i','n','h','\0'}; hoặc
char Name[]="Minh"; đều cho kết quả giống nhau.

Ví dụ 3-13. Viết chương trình nhập ma trận vuông *A* cấp *n*, các phần tử là số nguyên. Đưa ma trận tam giác dưới và ma trận tam giác trên của *A* ra màn hình. Các dữ liệu được nhập từ bàn phím.

```
/* ****************************************************************************** */
#include "stdio.h"
#include "conio.h"
/* ****************************************************************************** */
int main()
```

```
{
    int a[30][30], i, j, n;
    clrscr();
    printf("\nHay nhap so hang (cot) n= ");
    scanf("%d", &n);
    printf("\nVao ma tran A\n");
    for(i=0; i<n; ++i)
        for(j=0; j<n; ++j)
        {
            printf("A[%d][%d]= ", i, j);
            scanf("%d", &a[i][j]); /* vì đây là ma trận nguyên */
        }
    clrscr();
    printf("\nMa tran tam giac tren (gom ca duong cheo chinh) nhu sau \n");
    for(i=0; i<n; ++i)
    {
        for(j=i; j<n; ++j) /* chỉ xét các phần tử trên đường chéo chính */
        {
            gotoxy(8*(j+1), 3+i);
            printf("%7d", a[i][j]);
        }
        printf("\n"); /* hết một hàng của ma trận */
    }
    printf("\nMa tran tam giac duoi (gom ca duong cheo chinh) la:\n\n");
    for(i=0; i<n; ++i)
    {
        for(j=0; j<= i; ++j) /* chỉ xét các phần tử dưới đường chéo chính*/
            printf("%7d", a[i][j]);
        printf("\n"); /* sang hàng tiếp theo */
    }
    getch();
    return 0;
}
/* ****************************************************************************** */
```

Ví dụ 3-14. Viết chương trình nhập từ bàn phím ma trận *T* kích thước *m x n* (*m<=20, n<=35*). Tính ma trận *S* cùng kích thước sao cho: $S_{i,j} = \begin{cases} 1 \text{ nếu } u_{i,j} > 0 \\ 0 \text{ nếu } u_{i,j} = 0 \\ -1 \text{ nếu } u_{i,j} < 0 \end{cases}$

Đưa ra màn hình ma trận *S* theo đúng hàng, cột.

```
/* ****************************************************************************** */
#include "stdio.h"
#include "conio.h"
/* ****************************************************************************** */
int main()
{
    float T[20][35], a, b;
    int S[20][35], i, j, m, n;
    clrscr();
```

```
    printf("\nNhap kich thuoc mang m,n = ");
    scanf("%d%d", &m, &n);
    printf("\nNhap mang T:\n");
    for(i=0; i<m; ++i)
        for(j=0; j<n; ++j)
        {
                printf("\nT[%d][%d]= ", i, j);
                scanf("%f", (float*)T+ i*35 + j);
                if(T[i][j]>0)
                        S[i][j]= 1;
                else if (T[i][j]<0)
                        S[i][j]= -1;
                else
                        S[i][j]= 0;
        }
    printf("\nKet qua S la:\n");
    for(i=0; i<m;++i)
    {
            printf("\n");
            for(j=0; j<n; ++j)
                printf("%3d", S[i][j]);
    }
    getch();
    return 0;
}
/* ******************************************************************************************* */
```

4. Hạn chế của việc sử dụng mảng

Xét bài toán cộng hai đa thức bậc ***n*** hai ẩn số ***x*** và ***y***. Nhận xét rằng, khi cộng hai đa thức ta phải tìm kiếm các số hạng cùng bậc của ***x*** và ***y*** rồi cộng các hệ số lại với nhau. Như vậy, để việc cộng hai đa thức có thể làm việc một cách hiệu quả, ta phải có cách biểu diễn nào đó để phân biệt được các biến, hệ số và số mũ trong mỗi đa thức thành phần. Có nhiều cách biểu diễn khác nhau cho đa thức, trong đó có một cách tương đối hiệu quả và dễ sử dụng đó là dùng ma trận để biểu diễn theo nguyên tắc sau: mỗi ma trận vuông cấp ***n*** dùng để biểu diễn cho một đa thức cấp $(n-1)$ của ***x*** và ***y***, trong đó mỗi phần tử ở hàng ***i*** cột ***j*** của ma trận sẽ lưu trữ hệ số của phần tử $x^i y^j$ của đa thức. Ví dụ các ma trận sau dùng để biểu diễn cho các đa thức:

$P(x,y)=8x^4 + 6y^3x^2 + 3y^2x^3 + xy - 1$
$Q(x,y)=4x^4 + 12y^2x^3 + 7y^2x^2 + xy + 11$

P(x,y) + **Q(x,y)**

$$
\begin{array}{c|ccccc}
 & 0 & 1 & 2 & 3 & 4 \\
\hline
0 & -1 & 0 & 0 & 0 & 0 \\
1 & 0 & 1 & 0 & 0 & 0 \\
2 & 0 & 0 & 0 & 6 & 0 \\
3 & 0 & 0 & 3 & 0 & 0 \\
4 & 8 & 0 & 0 & 0 & 0
\end{array}
\quad + \quad
\begin{array}{c|ccccc}
 & 0 & 1 & 2 & 3 & 4 \\
\hline
0 & 11 & 0 & 0 & 0 & 0 \\
1 & 0 & 1 & 0 & 0 & 0 \\
2 & 0 & 0 & 7 & 0 & 0 \\
3 & 0 & 0 & 12 & 0 & 0 \\
4 & 4 & 0 & 0 & 0 & 0
\end{array}
$$

Hình 3.4. Mô tả phép cộng hai đa thức hai ẩn dùng ma trận.

Bài toán cộng hai đa thức giờ đây đã chuyển thành bài toán cộng hai ma trận cùng cấp (*có thể mở rộng bài toán thành trừ, nhân, chia... hai đa thức*) và kết quả phép cộng như sau (*hình 3.5.*).

$$C(x,y) = \begin{array}{c} \\ 0 \\ 1 \\ 2 \\ 3 \\ 4 \end{array} \begin{array}{c} \begin{array}{ccccc} 0 & 1 & 2 & 3 & 4 \end{array} \\ \begin{pmatrix} 10 & 0 & 0 & 0 & 0 \\ 0 & 2 & 0 & 0 & 0 \\ 0 & 0 & 7 & 6 & 0 \\ 0 & 0 & 15 & 0 & 0 \\ 12 & 0 & 0 & 0 & 0 \end{pmatrix} \end{array}$$

Hình 3.5. Kết quả phép cộng hai đa thức dưới dạng ma trận.

Từ đó: $C(x,y)=P(x,y)+Q(x,y)= 12x^4+6y^3x^2+15y^2x^3+7y^2x^2+2xy+10$

Ta nhận thấy, việc dùng mảng để biểu diễn cho đa thức như trên đã làm giảm đi rất nhiều độ phức tạp của bài toán. Tuy nhiên, có những hạn chế dễ nhận thấy như sau:

- Mỗi mảng cấp ***nxn*** chỉ có thể biểu diễn được cho một đa thức cấp (***n-1***) của ***x*** và ***y***, do đó hạn chế *phạm vi xử lí*.

- Nếu các đa thức được biểu diễn là không đầy đủ (*ví dụ đa thức* $P(x,y) = x^{1000} + y$), khi đó ta phải dùng một ma trận *1001 x 1001* phần tử để lưu trữ hai hệ số của ***P(x,y)***, còn tất cả (***1001 x 1001 - 2***) phần tử còn lại đều có giá trị bằng không và không được sử dụng đến. Điều này sẽ gây ra một sự lãng phí bộ nhớ do tính cố định của mảng. Để có thể khắc phục được các nhược điểm của mảng, chúng ta sẽ tiếp tục nghiên cứu một cấu trúc dữ liệu đặc trưng tiếp theo trong mục *3.3.4*, đó là cấu trúc dữ liệu kiểu danh sách móc nối.

5. Chuỗi (String) và xử lí chuỗi

Chuỗi hay *xâu kí tự* là một mảng kiểu *char* kết thúc bằng kí tự *'\0'*.

Cũng giống như tên mảng, tên chuỗi cũng là một hằng con trỏ chứa địa chỉ của kí tự đầu tiên trong chuỗi.

Ví dụ, để nhập từ bàn phím tên của một người ta cần dùng một mảng kiểu kí tự với đoạn chương trình sau:

```
char Ten[30];
printf("\nHay nhap ho va ten:");
gets(Ten);
```

Ví dụ 3-15. Viết chương trình nhập từ bàn phím một câu (*ít hơn 80 kí tự*) rồi đếm xem trong câu vừa nhập có mấy *từ* (*xâu khác rỗng không chứa dấu cách*)?

Giải. Để viết được chương trình ta cần dùng thêm hàm:

int isspace(int KyTu) khai báo trong ***ctype.h*** dùng để kiểm tra kí tự ***KyTu*** có phải là dấu cách hay không? hàm trả về giá trị 1 (*đúng*) nếu KyTu là dấu cách và trả về 0 nếu ngược lại, và hàm ***int strlen(char *s)*** khai báo trong ***string.h*** dùng để tính độ dài của chuỗi ***s***.

```
/* ************************************************************************************ */
#include "stdio.h"
#include "conio.h"
```

```
#include "ctype.h"
#include "string.h"
/* ******************************************************************************** */
int main()
{
     char Cau[81];
     int L, i, Dem=0;
     clrscr();
     printf("\nHay nhap mot cau khong qua 80 ký tu\n");
     gets(Cau);
     L= strlen(Cau);
     Cau[L]= ' '; /* dùng kĩ thuật đặt lính canh để đánh dấu sự kết thúc tìm */
     Cau[L+1]= '\0' ; /* Đánh dấu lại sự kết thúc của chuỗi */
     for(i=0; i<=L; ++i)
         if(!isspace(Cau[i])&&isspace(Cau[i+1]))
               Dem++;
     printf("\n\nSo tu trong cau la: %d\n", Dem);
     getch();
     return 0;
}
/* ******************************************************************************** */
```

Ví dụ 3-16. Viết chương trình nhập từ bàn phím một xâu kí tự sau đó nhập vào một kí tự bất kì rồi đếm số lượng kí tự này có trong xâu vừa nhập, thông báo kết quả ra màn hình.

```
/* ******************************************************************************** */
#include "stdio.h"
#include "conio.h"
#include "string.h"
/* ******************************************************************************** */
int main()
{
     char Xau[80], Ch;
     int L, i, Dem=0;
     clrscr();
     printf("\nHay nhap mot xau khong qua 80 ky tu\n");
     gets(Xau);
     printf("\nHay nhap  ky tu can dem\n");
     scanf("%c", &Ch);
     L=strlen(Xau);
     for(i=0; i<L; ++i)
         if(Xau[i]==Ch)
               Dem++;
     printf("\n\nSo ky tu  %c  trong xau la: %d\n", Ch, Dem);
     getch();
     return 0;
}
/* ******************************************************************************** */
```

Ví dụ 3-17. Viết chương trình liệt kê một bảng *MENU*, dữ liệu nhập vào từ bàn phím gồm:

- Tên tiêu đề *MENU* (*không quá 30 kí tự*).
- Số lượng các *MENU* (*n* <20).
- Nội dung từng *MENU* (*không quá 30 kí tự*).

Yêu cầu chương trình phải hiện lên màn hình các nội dung nằm trong một khung chữ nhật. Kích thước của khung chữ nhật phụ thuộc vào *n* và độ dài các xâu.

```
/* ************************************************************************************** */
#include "stdio.h"
#include "conio.h"
#include "string.h"
/* ************************************************************************************** */
int main()
{
    char TieuDe[30], NoiDung[20][30];
    int Max, x, y, n, i;
    clrscr();
    printf("\nHay nhap Tieu de cho MENU\n");
    gets(TieuDe);
    Max= strlen(TieuDe);
    printf("\nHay nhap  so luong MENU n= ");
    scanf("%d", &n);
    fflush(stdin);/* Làm sạch dòng nhập, loại bỏ kí tự xuống dòng do hàm
    scanf để lại. Nếu không câu lệnh gets ở sau sẽ bị trôi */
    for(i=0; i<n; ++i)
    {
        printf("\nNoi dung MENU %d la: ", i);
        gets(NoiDung + i);
        if(Max< strlen(NoiDung + i))
            Max= strlen(NoiDung + i);
    }
    clrscr();
    Max+= 4; /* độ rộng của khung */
    x=(int)((80- strlen(TieuDe))/2+0.5);
    y=2;
    gotoxy(x,y);
    puts(TieuDe); /*Đưa chuỗi TieuDe ra màn hình sau đó xuống dòng */
    gotoxy(x, y+1);
    for(i=1; i<=strlen(TieuDe)+1; ++i)
        putchar('=');/* vẽ khung dưới tiêu đề */
    x=(int)((80-Max)/2);
    y=4;
    gotoxy(x, y); /* chuẩn bị hiển thị nội dung của từng mục Menu */
    for(i=1; i<=Max+5; ++i)
        putchar('*'); /* vẽ khung bao trên */
    for(i=0; i<n; ++i)
    {
        gotoxy(x, y+i+1);
        printf("*  %d. %s", i, NoiDung + i);
        gotoxy(x+Max+4, y+i+1);
        puts("*"); /* vẽ khung bên phải */
```

```
    }
    gotoxy(x, y+n+1);
    for(i=1; i<=Max+5; ++i)
        putchar('*'); /* vẽ khung dưới */
    getch();
    return 0;
}
/* ****************************************************************************** */
```

Bài tập: Chạy và phân tích hoạt động của chương trình trên.

Ví dụ 3-18. Viết chương trình nhập vào từ bàn phím một văn bản sau đó chèn vào văn bản đó tại vị trí thứ ***i*** một xâu mới. Đưa kết quả ra màn hình.

```
/* ****************************************************************************** */
#include "stdio.h"
#include "conio.h"
#include "string.h"
/* ****************************************************************************** */
int main()
{
    char VanBan[80], Xau[30];
    int i;
    clrscr();
    puts("Hay nhap mot van ban:");
    gets(VanBan);
    puts("Hay nhap chuoi can chen");
    gets(Xau);
    puts("Hay nhap vi tri chen");
    scanf("%d", &i);
    clrscr();
    puts("Xau ban dau la:");
    puts(VanBan);
    if(i> strlen(VanBan))
    {
        strcat(VanBan, Xau);/* Ghép chuỗi Xau vào sau chuỗi VanBan*/
    }
    else
    {
        char Luu[80];
        /* Copy các kí tự còn lại trong xâu VanBan bắt đầu từ vị trí thứ i
        của xâu */
        strncpy(Luu, VanBan+i, strlen(VanBan)-i+1);
        /* Gắn thêm dấu cách vào sau i kí tự đầu tiên của xâu VanBan
        trước khi ghép, sau đó cắt xâu tại vị trí đó bằng kí tự kết thúc*/
        VanBan[i]= ' ';
        VanBan[i+1]= '\0';
        /* Ghép Xau vào sau i kí tự của xâu VanBan */
        strcat(VanBan, Xau);
        /* Tách các đoạn ghép bằng dấu cách */
        i= strlen(VanBan);
        VanBan[i]= ' ';
        VanBan[i+1]= '\0';
```

```
        /* Ghép phần còn lại của xâu VanBan vào chuỗi */
        strcat(VanBan, Luu);
    }
    puts("Van ban sau khi chen la");
    puts(VanBan);
    getch();
    return 0;
}
/* ************************************************************************************* */
```

Nhận xét:

Chương trình trên đây đã bao gồm nhiều thao tác đặc trưng cho việc xử lí xâu kí tự trong ngôn ngữ lập trình C như việc ghép xâu (dùng hàm ***char* strcat(char *ChuoiNhan, char *ChuoiGhep)***), chép ***n*** kí tự đầu tiên từ xâu nguồn sang xâu đích (dùng hàm ***char* strncpy(char * XauDich, char* XauNguon, n)***)...(*xem thêm phụ lục III*) và đặc biệt là cách điều khiển sự kết thúc của mỗi xâu.

Bài tập: Xét một đoạn của chương trình trên

```
strncpy(Luu, VanBan+i, strlen(VanBan)-i+1);
VanBan[i]=' ';
VanBan[i+1]='\0';
```

Điều gì sẽ xảy ra khi hai câu lệnh đứng sau cùng vắng mặt? Hãy chạy thử chương trình trong trường hợp đó và so sánh với nhận xét của bản thân!

Chú ý:

Ta không thể đưa *trực tiếp* một hằng kí tự (*một xâu trong dấu* "") vào một *biến xâu* được. Ví dụ các câu lệnh sau đây là không hợp lệ:

```
char HoVaTen[30];
HoVaTen= "Nguyen Van A"; /* Sai */
```

Do ***HoVaTen*** là một hằng con trỏ chứa địa chỉ đầu của vùng nhớ cấp phát cho biến xâu, "*Nguyen Van A*" cũng là một hằng con trỏ chứa địa chỉ của vùng nhớ lưu chuỗi "*Nguyen Van A*". Ta không thể gán một hằng con trỏ này cho một hằng con trỏ khác được. Để làm được điều đó ta có thể gán thông qua cơ chế khởi đầu, thông qua hàm ***scanf***, thông qua con trỏ hoặc dùng các hàm thao tác trên chuỗi như ví dụ sau:

```
char HoVaTen[30];
strcpy(HoVaTen, "Nguyen Van A");
```

6. Liên hệ giữa con trỏ và mảng

Trong ngôn ngữ lập trình C, con trỏ và mảng có mối liên hệ mật thiết với nhau. Ta đã biết rằng tên mảng là một hằng con trỏ chứa địa chỉ của phần tử đầu tiên trong mảng, do đó các câu lệnh sau đây là hoàn toàn hợp lệ:

```
int MangNguyen[30], *Pi;
float MangThuc[30], *Pf;
Pi= MangNguyen; /* Cho Pi trỏ tới vùng nhớ của mảng MangNguyen */
Px= MangThuc; /* Cho Pf trỏ tới vùng nhớ của mảng MangThuc */
```

Sau các câu lệnh trên thì :

Pi + i sẽ trỏ tới phần tử *MangNguyen[i]*.
Pf + i sẽ trỏ tới phần tử *MangThuc[i]*.

Và để truy nhập đến các phần tử này, các câu lệnh sau là *tương đương* nhau:

MangNguyen[i] =10; *(Pi + i)=10; *(MangNguyen + i)=10; và Pi[i]=10; hoặc
MangThuc[i]=2.1; *(Pf + i)=2.1; *(MangThuc + i)=2.1; và Pf[i]=2.1;

Tuy nhiên vấn đề trở nên phức tạp hơn khi sử dụng con trỏ với mảng nhiều chiều. Ví dụ với các câu lệnh:

```
float *Pf, MangThuc[20][30];
Pf = (float*) MangThuc;
```

Thì những câu lệnh sau đây:

MangThuc[i][j] [22] =3.5; Pf[i][j]=3.5; *(Pf + i*30 + j)=3.5;
((float)MangThuc + i*30 +j)=3.5; sẽ tương đương với nhau.

7. Mảng con trỏ

Mảng con trỏ là mảng đặc biệt mà mỗi phần tử của nó là một con trỏ dùng để chứa địa chỉ của một biến nào đó và nó có đầy đủ các tính chất của một mảng. Mảng con trỏ được khai báo theo mẫu sau:

Kiểu *TênMảng[KíchThướcMảng];

Ví dụ 3-19. Câu lệnh ***double *MangConTrof[100]*** sẽ khai báo một mảng gồm *100* phần tử, mỗi phần tử là một con trỏ kiểu ***double*** có thể dùng chứa kiểu địa chỉ *double*.

Chú ý, cần phân biệt khai báo trên với khai báo double (*Mang)[100]; dùng để khai báo ra một con trỏ có thể chứa được *kiểu địa chỉ double[100]*.

3.2.3. Cấu trúc (*Struct*) và hợp (*Union*)

Để lưu trữ và xử lí thông tin trong máy tính ta cần dùng đến các biến và mảng, tuy nhiên mỗi biến lại chỉ chứa được một giá trị cho một kiểu xác định. Còn mảng có thể xem là tập hợp của nhiều biến có cùng một kiểu giá trị và được biểu thị bằng một tên. Trong thực tế, việc lưu trữ và xử lí thông tin không phải đơn thuần chỉ thao tác trên một giá trị hoặc tập các giá trị cùng kiểu, mà đòi hỏi có sự tổ hợp của nhiều kiểu dữ liệu thành phần trong cùng một đối tượng xử lí. Chẳng hạn, để xử lí thông tin liên quan đến một đối tượng *nhân viên* ta cần các kiểu dữ liệu như mã số nhân viên, tên, địa chỉ, ngày sinh, quê quán, mức lương... và các dữ liệu này phải được *xử lí thống nhất* để đảm bảo tính toàn vẹn dữ liệu. Các đối tượng dữ liệu loại này rất đa dạng, phong phú và thường do người lập trình tự định nghĩa ra tùy theo nhu cầu của họ. Các đối tượng dữ liệu như vậy trong ngôn ngữ lập trình C người ta gọi là các ***cấu trúc*** (*struct – tương tự bản ghi trong Pascal*) và ***hợp*** (*union*). Mặc dù cấu trúc và hợp có nhiều điểm tương đồng, song chúng cũng có những đặc trưng riêng và được sử dụng cho các mục đích khác nhau. Trong phần này ta sẽ nghiên cứu kĩ đặc điểm của từng loại.

---

[22] Với 0<=i <=19 và 0 <=j <= 29

1. Cấu trúc (*Struct*)

Cấu trúc là một kiểu dữ liệu do người sử dụng tự định nghĩa ra bao gồm nhiều thành phần, mỗi thành phần có thể là một biến, mảng hay con trỏ có kiểu đã được định nghĩa sẵn trong ngôn ngữ (*như int, float, double...*) hoặc lại là một *cấu trúc* (*hay hợp*) nào đó.

Cú pháp khai báo:

```
struct [TênKiểuCấuTrúc]
{
        /* Khai báo các thành phần dữ liệu ở đây;*/
} [Danh sách biến];
```

Trong đó, ***struct*** là từ khóa dùng để định nghĩa cấu trúc. ***TênKiểuCấuTrúc*** là tên bất kì do người lập trình đặt ra theo quy tắc đặt tên trong ngôn ngữ lập trình C.

Ví dụ 3-20. Để định nghĩa ra cấu trúc dữ liệu dùng để lưu trữ các thông tin của thí sinh trong một kì tuyển sinh ta có thể viết như sau:

```
struct Date
{
        int Ngay;
        int Thang;
        int Nam;
};
struct ThiSinh
{
        char SoBaoDanh[10];
        char  HoVaTen[30];
        char QueQuan[50];
        struct Date  NgaySinh;
        float DiemToan;
        float DiemLy;
        float DiemHoa;
} TS1, TS2, *DS;
struct ThiSinh TS3,TS4, DanhSach[50], *Pts;
```

Một cấu trúc sau khi định nghĩa xong ta có thể sử dụng nó để khai báo cho các biến, mảng và con trỏ. *Có hai cách* để khai báo biến, mảng hay con trỏ kiểu cấu trúc, đó là *khai báo trực tiếp* trong lúc định nghĩa (*như biến TS1, TS2 và con trỏ DS ở trên*) và khai báo sau khi đã định nghĩa (*như biến TS3, TS4, mảng DanhSach[50] và con trỏ Pts hoặc biến NgaySinh ở trên*).

Một biến, mảng hay con trỏ cấu trúc sau khi khai báo sẽ ***có đầy đủ mọi tính chất*** như các biến, mảng hay con trỏ thông thường. Ngoài ra nó còn có thêm một vài đặc trưng khác như sau:

a) Các thao tác trên một cấu trúc

- Truy nhập đến các thành phần của cấu trúc : Để truy nhập đến một thành phần của một biến hay phần tử thứ *i* mảng kiểu cấu trúc ta có thể viết như sau:

TênBiếnCấuTrúc.TênThànhPhần và

TênMảngCấuTrúc[i]. TênThànhPhần

Còn để truy nhập đến các thành phần của một biến cấu trúc thông qua con trỏ ta viết như sau:

TênConTrỏ->TênThànhPhần hoặc
(*TênConTrỏ).TênThànhPhần

- ***Phép gán trên các cấu trúc***: Với hai biến cấu trúc cùng kiểu *Bien1* và *Bien2*, ta hoàn toàn có thể gán các giá trị tương ứng của *Bien1* vào *Bien2* như sau:

Bien2=Bien1;

Ví dụ 3-21. Đoạn chương trình sau đây sẽ minh họa việc gán các giá trị cho biến, mảng cấu trúc (*gán giá trị cho các thành phần của cấu trúc*):

Pts=&TS1; /* Con trỏ *Pts* trỏ đến biến cấu trúc *TS1**/
DS=DanhSach; /*Con trỏ *DS* trỏ đến phần tử đầu tiên của mảng *DanhSach**/
TS2.NgaySinh.Ngay=11;/*Đưa giá trị *11* vào thành phần ngày của biến TS1*/
Pts->DiemToan=10; /*Đưa giá trị 10 vào thành phần DiemToan của biến TS1 thông qua con trỏ Pts*/
DS->DiemLy=9; /*Đưa giá trị 9 vào thành phần DiemLy của phần tử DanhSach[0] của mảng cấu trúc*/
(DS+2)->DiemHoa=7; /*Đưa giá trị 7 vào thành phần DiemHoa của phần tử DanhSach[2] của mảng cấu trúc*/
DS[4].DiemToan=5;/*Đưa giá trị 5 vào thành phần DiemToan của phần tử mảng DanhSach[4]*/
DanhSach[4].DiemLy=6;/* Đưa giá trị 6 vào thành phần DiemLy của phần tử mảng DanhSach[4]*/
*(DS+4).DiemHoa=3;/*Đưa giá trị 3 vào thành phần DiemHoa của phần tử mảng DanhSach[4]*/
DS[5]=*Pts; /* Gán nội dung của biến cấu trúc TS1 cho phần tử mảng DanhSach[5]*/ [23]
*(DS+6)=TS2; /*Gán nội dung của biến cấu trúc TS2 cho phần tử mảng DanhSach[6]*/
TS4=TS3; /*Gán nội dung của biến cấu trúc TS3 cho biến cấu trúc TS4*/

b) Các thành phần kiểu nhóm Bit

Để tiết kiệm bộ nhớ đối với các thành phần nguyên (***signed*** hoặc ***unsigned***) khi biểu diễn một miền giá trị nhỏ (*ví dụ như tuổi thường có giá trị chỉ từ 0 đến 100*) trong ngôn ngữ lập trình C cho phép khai thác đến *từng Bit như là các thành phần riêng của một cấu trúc*. Một thành phần như vậy gọi là thành phần *kiểu nhóm Bit*. Ví dụ như sau:

```
struct SinhNhat
{
	unsigned int Ngay: 5;
	unsigned int Thang: 4;
	unsigned int  Tuoi:7;
};
```

[23] Mỗi phép gán trên sẽ tương ứng với một dãy phép gán các thành phần tương ứng của các biến cấu trúc.

Sẽ định nghĩa ra một cấu trúc lưu giữ thông tin sinh nhật của một người. Trong đó thành phần *Ngay* sẽ chiếm *5 bits* (*biểu diễn được từ 0 cho đến 31*), thành phần *Thang* chiếm *4 bits* (*biểu diễn được từ 0 cho đến 15*) và thành phần *Tuoi* chiếm *7 bits* (*biểu diễn được từ 0 cho đến 127*), do đó kích thước của cấu trúc này chỉ là *16 bits* (*2 bytes*) thay vì *6 bytes* nếu định nghĩa theo cách thông thường. Việc truy nhập đến các thành phần nhóm bit cũng tương tự như các thành phần khác (*xem thêm phần Union để hiểu rõ hơn về ứng dụng của các thành phần nhóm bit*).

Chú ý:

- Khi sử dụng các thành phần kiểu nhóm bit, độ dài tối đa của mỗi thành phần là 16.

- Không cho phép lấy địa chỉ thành phần kiểu nhóm bit.

- Không thể xây dựng các mảng kiểu nhóm bit.

- Không thể trả về từ hàm bằng một thành phần kiểu nhóm bit.

- Khi muốn bỏ qua một số bit thì ta bỏ trống tên trường.

c) Khởi đầu cho một cấu trúc

Có thể khởi đầu (*một lần vào lúc dịch chương trình*) cho cấu trúc ngoài, cấu trúc tĩnh, mảng cấu trúc ngoài và mảng cấu trúc tĩnh bằng cách viết vào sau khai báo của chúng một danh sách tương ứng các giá trị cho các thành phần (*các thành phần phân tách nhau bởi dấu phẩy*). Ví dụ với cấu trúc ***ThiSinh*** ở trên ta có thể khởi đầu như sau:

```
struct ThiSinh TS={
                    "BKA2003",
                    "Nguyen Van A",
                    "Ba Dinh – Ha Noi",
                    {11,12,1967},
                    10,
                    8,
                    6
                };
```

d) Cấu trúc tự trỏ

Cấu trúc tự trỏ là một dạng cấu trúc đặc biệt có chứa một thành phần là con trỏ *trỏ đến chính nó* (*trong phần 3.4 sẽ tìm hiểu kĩ hơn cách sử dụng của cấu trúc tự trỏ trong danh sách móc nối*). Một cấu trúc tự trỏ có thể được định nghĩa theo cú pháp sau:

```
struct TênCấuTrúc
{
      Khai báo các thành phần của cấu trúc;
      struct TênCấuTrúc *Tiep; /*Con trỏ dùng để trỏ đến các biến cấu trúc
      khác cùng kiểu */
}     [Danh sách biến];
```

Ví dụ 3-22. Ví dụ về cấu trúc tự trỏ.

```
struct Date
{
       int Ngay;
```

```
        int Thang;
        int Nam;
    };
    struct ThiSinh
    {
        char SoBaoDanh[10];
        char  HoVaTen[30];
        char QueQuan[50];
        struct Date  NgaySinh;
        float DiemToan;
        float DiemLy;
        float DiemHoa;
        struct ThiSinh *Tiep;
    } TS1,TS2;
```

Khi đó ta hoàn toàn có thể thực hiện được các câu lệnh dưới đây:

TS1.Tiep= &TS2;/*Thành phần **Tiep** của biến cấu trúc **TS1** sẽ trỏ đến cấu trúc **TS2***/

TS1.Tiep->DiemLy=8; /*Đưa giá trị 8 vào thành phần **DiemLy** của biến cấu trúc **TS2** */

Chú ý:

- Nếu ta đặt từ khóa ***typedef*** trước định nghĩa của một cấu trúc, thì khi khai báo một biến cho cấu trúc đó ta không cần sử dụng từ khóa ***struct*** vào trước tên của cấu trúc nữa.

- Trong định nghĩa của một cấu trúc có thể vắng mặt tên *kiểu cấu trúc*, nhưng trong chương trình ta sẽ không thể khai báo thêm các biến cấu trúc được nữa (*trừ những biến đã được khai báo trong lúc định nghĩa*).

- Để tránh dài dòng khi truy nhập vào các thành phần của cấu trúc ta có thể dùng lệnh *#define* như sau:

```
#define TS  TS1.NgaySinh
...
printf("\nHay nhap ngay, thang nam sinh cho thi sinh TS1");
scanf("%d",&TS.Ngay);
scanf("%d",&TS.Thang);
scanf("%d",&TS.Nam);
...
```

- Phép lấy địa chỉ chỉ thực hiện tốt đối với các thành phần nguyên của một biến cấu trúc (*như thành phần Ngay, Thang hoặc Nam trên đây*). Đối với các thành phần không nguyên việc làm đó có thể dẫn đến treo máy. Do đó, trong trường hợp này trước tiên ta nên thao tác trên *biến trung gian*, sau đó mới gán giá trị đó cho thành phần của cấu trúc. Đoạn chương trình sau có thể minh họa rõ hơn điều đó.

```
...
float TrungGian;
printf("\nHay nhap diem toan cho thi sinh TS1");
scanf("%f", &TrungGian);
TS1.DiemToan=TrungGian;
...
```

Ví dụ 3-23. Viết chương trình nhập vào một danh sách *n* sinh viên lớp Z gồm họ tên, năm sinh, điểm thi vào trường (*điểm toán + điểm lí + điểm hóa*). Đưa danh sách này ra màn hình.

```
/* ************************************************************************************ */
#include "stdio.h"
#include "conio.h"
struct SinhVien
{
    char HoTen[30] ;
    int NamSinh;
    float Diem;
};
/* ************************************************************************************ */
int main()
{
    struct SinhVien DanhSach[100];
    int n, i;
    float Tam;
    clrscr();
    printf("\nSo luong Sinh vien= ");
    scanf("%d", &n);
    printf("\nNhap du lieu cho lop Z \n\n");
    for(i=0; i<n; ++i)
    {
        printf("\nNhap ho va ten cua Sinh vien thu %d \n", i);
        fflush(stdin);
        gets(DanhSach[i].HoTen);
        printf("\nSinh nam: ");
        scanf("%d", & DanhSach[i].NamSinh);
        printf("\nDiem thi vao truong: ");
        scanf("%f", &Tam);
        DanhSach[i].Diem=Tam;
    }
    clrscr();
    gotoxy(10, 2); printf("DANH SACH SINH VIEN LOP Z");
    gotoxy(10, 3); printf("--------------------------------------");
    gotoxy(5, 6);   printf("TT");
    gotoxy(10, 6); printf("Ho va Ten");
    gotoxy(40, 6); printf("Nam sinh");
    gotoxy(50, 6); printf("Diem thi vao truong");
    for(i=0;i<n;++i)
    {
        gotoxy(5, 8+i+1); printf("%d",i);
        gotoxy(10, 8+i+1); printf("%s",DanhSach[i].HoTen);
        gotoxy(40, 8+i+1); printf("%d",DanhSach[i].NamSinh);
        gotoxy(50, 8+i+1); printf("%3.1f",DanhSach[i].Diem);
    }
    getch();
    return 0;
}
```

Bài tập:

Viết lại chương trình trên có dùng lệnh ***#define*** để đơn giản việc truy nhập vào biến cấu trúc.

Ví dụ 3-24. Viết chương trình thực hiện các công việc sau:

1. Nhập thông tin từ bàn phím về tình hình thời tiết trong ngày của khu vực. Mỗi bản tin là một cấu trúc gồm các trường: ngày, tháng, năm, địa điểm đo (*xâu kí tự độ dài không quá 35*), lượng mưa, nhiệt độ. Số lượng bản tin không biết trước. Dấu hiệu kết thúc nhập là bản tin có trường ngày, tháng và năm bằng không.

2. Hãy tìm xem ngày nào và ở địa điểm nào có nhiệt độ cao nhất ? Đưa ra màn hình kết quả đó.

3. Đưa ra màn hình lượng mưa trung bình trong ngày của khu vực.

```
/* ******************************************************************************************* */
#include "stdio.h"
#include "conio.h"
struct Date
{
    unsigned Ngay: 5;
    unsigned Thang:4;
    unsigned Nam: 15;
};
struct ThoiTiet
{
    struct Date ThoiGian;
    char DiaDiem[35];
    float LuongMua;
    int NhietDo:8;
};
/* ******************************************************************************************* */
int main()
{
    struct ThoiTiet  BaoCao[1000], ThoiTietNgay;
    int n, i, NhietDoMax, DanhDau, xToaDo, yToaDo;
    float LuongMuaTB;
    clrscr(); n= -1;
    printf("\nNhap du lieu \n");
    do
    {
        int TG;
        float Tam;
        printf("Ngay  ");
        xToaDo=wherex()+3; /* Hàm này trả về tọa độ trục x của con trỏ
        màn hình ở vị trí hiện tại */
        yToaDo= wherey(); /* Hàm này trả về tọa độ trục y  của con trỏ
        màn hình ở vị trí hiện tại, cả hai hàm đều trong conio.h */
        scanf("%d", &TG); ThoiTietNgay.ThoiGian.Ngay=TG;
        /*Nhập cho một bản tin*/
        if(TG!=0) /* kiểm tra điều kiện kết thúc */
        {
```

```
            gotoxy(xToaDo, yToaDo); xToaDo+=10;
            printf("Thang "); scanf("%d",&TG);
            ThoiTietNgay.ThoiGian.Thang=TG;
            gotoxy(xToaDo, yToaDo); printf("Nam "); scanf("%d",&TG);
            ThoiTietNgay.ThoiGian.Nam=TG;
            printf("Dia diem "); fflush(stdin);
            gets(ThoiTietNgay.DiaDiem);
            printf("Luong mua "); scanf("%f",&Tam);
            ThoiTietNgay.LuongMua=Tam;
            printf("Nhiet do "); scanf("%d",&TG);
            ThoiTietNgay.NhietDo=TG;
            printf("*****************\n");
        }
        if(ThoiTietNgay.ThoiGian.Ngay!=0)
        {
            n++; /* đếm số bản ghi thực tế */
            BaoCao[n]=ThoiTietNgay;
        }
    }
    while (ThoiTietNgay.ThoiGian.Ngay!=0); /* kết thúc nhập */
    /* Tìm nhiệt độ cao nhất và lượng mưa trung bình */
    NhietDoMax = -128; DanhDau = 0; LuongMuaTB=0;
    for(i=0; i<=n; ++i)
    {
        if(NhietDoMax < BaoCao[i].NhietDo)
        {
            NhietDoMax = BaoCao[i].NhietDo;
            DanhDau= i; /* Đánh dấu bản tin có nhiệt độ cao nhất */
        }
        LuongMuaTB+=BaoCao[i].LuongMua; /* tính tổng lượng mưa */
    }
    clrscr();
    printf("\nNhiet do cao nhat quan sat duoc la: %3d", NhietDoMax);
    printf("\nDo duoc tai %s", BaoCao[DanhDau].DiaDiem);
    printf("\nTrong ngay  %d  thang  %d  nam  %d",
    BaoCao[DanhDau].ThoiGian.Ngay,
    BaoCao[DanhDau].ThoiGian.Thang,
    BaoCao[DanhDau].ThoiGian.Nam);
    printf("\nLuong mua trung binh cua khu vuc la:%10.2f",
    LuongMuaTB/(n+1));
    getch();
    return 0;
}
/* ************************************************************************************** */
```

Bài tập:

- Giải thích hoạt động của chương trình trên.

- Tại sao ta không nhập trực tiếp giá trị vào các thành phần Ngay, Thang và Nam của các biến cấu trúc mà phải nhập thông qua biến TG ?

- Viết lại chương trình với câu lệnh ***#define***.

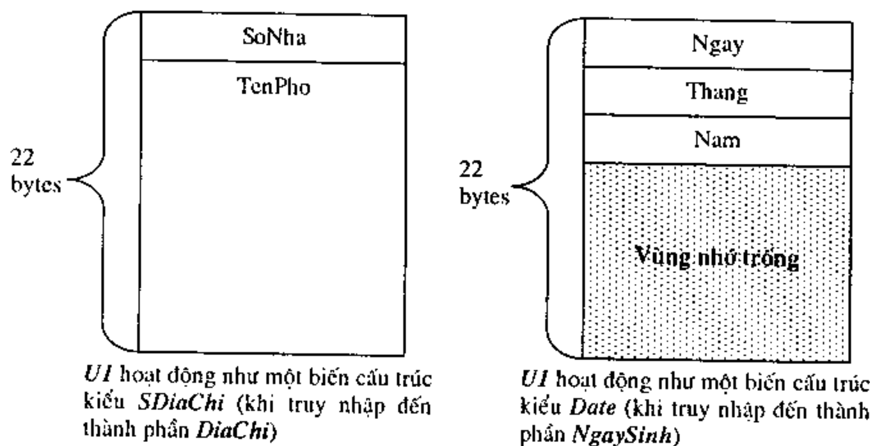
2. Hợp (*Union*)

Hợp là một loại ***cấu trúc đặc biệt*** được định nghĩa bằng từ khóa ***union*** (*thay cho struct*), có các thành phần chỉ ***dùng chung một vùng nhớ*** (*khác với cấu trúc, các thành phần của cấu trúc được cấp phát các vùng nhớ khác nhau và liên tiếp nhau trong bộ nhớ*) và kích thước của *hợp* sẽ bằng kích thước của thành phần lớn nhất. Nghĩa là, với *hợp* ta có thể khai báo ra các biến có khả năng chứa được nhiều kiểu dữ liệu khác nhau (*giống FOXPRO*).

Ví dụ 3-25. Để khai báo được một cấu trúc dữ liệu *hoặc* có thể chứa được địa chỉ, *hoặc* chứa được ngày tháng năm sinh của một người nào đó ta viết như sau:

```
struct Date
{
        unsigned Ngay;
        unsigned Thang;
        unsigned  Nam;
};
struct SDiaChi
{
        unsigned SoNha;
        char TenPho[20];
};
union DiaChi_NgaySinh
{
        struct Date NgaySinh;
        struct SDiaChi DiaChi;
} U1;
```

Thành phần ***NgaySinh*** của ***U1*** là một cấu trúc kiểu ***Date*** (*kích thước là 6 bytes*) và thành phần ***DiaChi*** của ***U1*** là một cấu trúc kiểu ***SDiaChi*** (*kích thước là 22 bytes*). Do đó kích thước của ***U1*** cũng là *22 bytes*. Cách làm việc của ***U1*** có thể mô tả qua hình vẽ sau:



U1 hoạt động như một biến cấu trúc kiểu ***SDiaChi*** (khi truy nhập đến thành phần ***DiaChi***)

U1 hoạt động như một biến cấu trúc kiểu ***Date*** (khi truy nhập đến thành phần ***NgaySinh***)

Hình 3.6. Sơ đồ mô tả cách làm việc của cấu trúc dạng hợp

Chú ý:

- Hợp có đầy đủ các tính chất của một cấu trúc.

- Tại một thời điểm ta không thể chứa dữ liệu tại tất cả các thành phần của một biến ***hợp*** được (*do các thành phần dùng chung vùng nhớ*).

- Ta có thể dùng thành phần kiểu ***nhóm bit*** và ***union*** để tách ra các bit của một từ theo ví dụ sau:

Ví dụ 3-26. Sử dụng union và nhóm bit để tách các từ:

```
union
{
	struct
	{
		unsigned a1;
		unsigned a2;
	} s;
	struct
	{
		unsigned n1:1;
		unsigned : 15; /* bỏ cách 15 bit tiếp theo */
		unsigned n2: 1;
		unsigned : 7; /* bỏ cách 7 bit tiếp theo */
		unsigned n3:8;
	} f;
} u;
```

Khi đó:

Các thành phần s và f của union dùng chung 4 bytes bộ nhớ và
u.f.n1 là bit 0 của u.s.a1;
u.f.n2 là bit 0 của u.s.a2;
u.f.n3 là byte cao của u.s.a2.

3.2.4. Danh sách (*List*)

1. Khái niệm và đặc trưng

Danh sách là một tập hợp ***có thứ tự*** gồm một số ***biến đổi*** các phần tử của ***một hoặc nhiều kiểu dữ liệu*** khác nhau. Các kiểu dữ liệu này thường là do người sử dụng tự định nghĩa ra (*các cấu trúc và hợp*).

Tập hợp những người đến mua hàng cho ta hình ảnh đặc trưng của một danh sách. Những người đến mua hàng sẽ xếp hàng theo một thứ tự nhất định (*ai đến trước sẽ được mua trước, ai đến sau sẽ được mua sau*...) và số lượng người mua hàng sẽ luôn luôn biến động trong các thời điểm khác nhau có lúc tăng lên (*do có người mới đến*), lúc giảm đi (*do có người chờ lâu đã bỏ về*).

Các đặc trưng cơ bản của danh sách

- Độ dài danh sách có thể *biến đổi*. Điều đó có nghĩa là khi sử dụng cấu trúc dữ liệu kiểu danh sách, ta luôn phải thực hiện thao tác *bổ sung hoặc loại bỏ* các phần tử của danh sách. Do đó, đối với việc lưu trữ các đối tượng mà có số lượng luôn luôn biến động thì cấu trúc dữ liệu kiểu danh sách thường được sử dụng.

- Khi danh sách không có *phần tử* nào người ta gọi là danh sách ***rỗng***.

- Các phần tử của một danh sách có thể cùng kiểu (*với danh sách được tổ chức theo kiểu kế tiếp hoặc móc nối*) hoặc có thể có nhiều hơn một kiểu (*với danh sách được tổ chức theo kiểu móc nối*). Điều đó có nghĩa là danh sách có thể lưu trữ giá trị cho các loại *đối tượng giống* hoặc *khác nhau*.

- Mỗi phần tử của danh sách được đặc trưng bởi các tham số là: ***giá trị, phần tử trước nó, phần tử sau nó*** (*ngoại trừ hai phần tử đặc biệt là phần tử đầu danh sách - là phần tử không có phần tử nào đứng trước và phần tử cuối danh sách - là phần tử không có phần tử nào đứng sau*). Việc truy nhập đến một phần tử ***i*** trong danh sách được thực hiện một cách ***gián tiếp*** thông qua việc duyệt tất cả (***i-1***) phần tử đứng trước nó (*khác với mảng*), bắt đầu từ phần tử đầu của danh sách. Do đó, thời gian truy nhập đến các phần tử trong danh sách là chậm hơn so với mảng và không đồng đều giữa các phần tử trong danh sách.

2. Cấp phát bộ nhớ động

Khi làm việc với danh sách, thông thường ta cần quản lí bộ nhớ một cách khá mềm dẻo đáp ứng nhu cầu biến động không ngừng của dữ liệu *trong lúc chạy chương trình*. Một cơ chế quản lí bộ nhớ linh hoạt như vậy được gọi là cấp phát bộ nhớ động. Các hàm dùng để cấp phát bộ nhớ động được khai báo trong thư viện ***alloc.h*** bao gồm:

void *calloc(unsigned n, unsigned size); Dùng để cấp phát vùng nhớ cho ***n*** đối tượng có kích thước ***size*** bytes. Nếu thành công hàm trả về địa chỉ đầu vùng nhớ được cấp, ngược lại hàm trả về giá trị NULL.

void* malloc(unsigned n); Dùng để cấp phát một vùng nhớ ***n*** byte. Nếu thành công hàm trả về địa chỉ đầu vùng nhớ được cấp, ngược lại trả về giá trị NULL.

void free(void *ptr); Dùng để giải phóng vùng nhớ đã cấp bằng cấp phát động do con trỏ ***ptr*** trỏ đến.

void* realloc(void *ptr, unsigned size); Dùng để thay đổi kích thước vùng nhớ đã được cấp phát động do con trỏ ***ptr*** trỏ đến với kích thước mới là ***size*** bytes. Các dữ liệu trên vùng nhớ cũ sẽ được chuyển tới vùng nhớ mới. Khi thành công hàm trả về địa chỉ của vùng nhớ mới, ngược lại hàm trả về giá trị *NULL*.

Chú ý:

Vì các hàm cấp phát bộ nhớ động đều làm việc với các con trỏ không kiểu, cho nên khi cấp phát cho biến kiểu nào ta cần ép về kiểu của biến đó.

Ví dụ 3-27. Đoạn chương trình sau sẽ cấp phát ra một vùng nhớ cho *10* số nguyên, sau đó gán các giá trị từ *0* đến *9* cho các số nguyên đó.

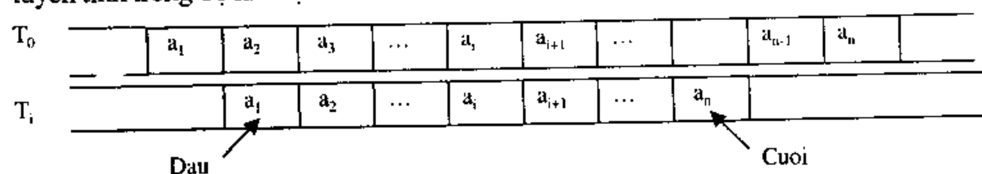
```
...
int *Pi, i;
Pi= (int*) malloc(10*sizeof(int)); /* Pi trỏ đến đầu vùng nhớ được cấp */
if( !Pi) /* nếu hết bộ nhớ */
{
        puts("\nKhong du bo nho ");
        exit(1); /* Làm cho chương trình kết thúc tức thì một cách bình
        thường */
```

```
}
for(i=0; i<10; ++i)
      Pi[i]= i; /* gán các giá trị từ 0 đến 9 cho các số nguyên vừa cấp */
...
```

3. Danh sách tuyến tính (*Linear list*)

Một danh sách mà các phần tử của nó được lưu trữ kế tiếp nhau trong bộ nhớ thì gọi là ***danh sách tuyến tính*** (*linear list*). Véctơ chính là trường hợp đặc biệt của danh sách tuyến tính tại một thời điểm xác định.

Như vậy, danh sách tuyến tính có thể coi là một *bộ có thứ tự* và luôn biến động các phần tử (a_1, a_2,... , a_n) cùng kiểu nào đó. Tệp (*file*) là một ví dụ điển hình về danh sách tuyến tính có kích thước lớn được lưu trữ ở bộ nhớ ngoài. Hình ảnh của danh sách tuyến tính trong bộ nhớ tại các thời điểm khác nhau có thể được mô tả như sau:



Hình 3.7. Hình ảnh của danh sách tuyến tính trong bộ nhớ tại các thời điểm.

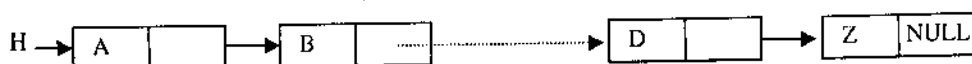
Do số lượng các phần tử của danh sách tuyến tính luôn biến động trong bộ nhớ, cho nên ta phải có một cơ chế nào đó để đánh dấu phần tử đầu tiên, phần tử cuối cùng của danh sách, cũng như phải nhận biết được trường hợp danh sách rỗng, nếu không ta sẽ không thể quản lí được danh sách. Có nhiều cách thức khác nhau để giải quyết cho vấn đề này. Một trong những cách hay được sử dụng đó là dùng ***hai biến trỏ*** để chứa địa chỉ của phần tử đầu và phần tử cuối của danh sách (*con trỏ **Dau** và con trỏ **Cuoi** trong hình vẽ*), mọi thao tác trên danh sách đều được thực hiện thông qua hai biến trỏ này. Thao tác duyệt toàn bộ danh sách sẽ phải được tiến hành thông qua một con trỏ khác, con trỏ này sẽ di chuyển từ đầu đến cuối danh sách. Thao tác bổ sung một phần tử vào danh sách sẽ được tiến hành bằng cách dãn các phần tử để lấy chỗ chèn. Ngược lại, phép loại bỏ một phần tử ra khỏi danh sách sẽ được tiến hành bằng cách dồn các phần tử lại để lấp đầy chỗ trống... Khi biến trỏ ***Dau*** có giá trị bằng biến trỏ ***Cuoi*** thì danh sách đã cho là rỗng.

Danh sách tuyến tính thường được dùng để cài đặt cho hai kiểu cấu trúc dữ liệu đặc biệt đó là ngăn xếp (*stack*) và hàng đợi (*queue*), hai cấu trúc này sẽ được xem xét kĩ hơn trong mục *3.3.5* và *3.3.6*.

4. Danh sách móc nối (*Linked list*)

a) Khái niệm

Danh sách móc nối là một loại danh sách mà các phần tử của nó (còn gọi là mục tin hay nút) được lưu trữ rải rác khắp nơi trong bộ nhớ, được nối kết với nhau theo một thứ tự nhất định nhờ vào các vùng dữ liệu đặc biệt gọi là vùng liên kết.



Hình 3.8. Hình ảnh của danh sách móc nối trong bộ nhớ.

Mỗi một phần tử của danh sách móc nối sẽ là một biến kiểu ***Cấu trúc tự trỏ*** hoặc kiểu ***Cấu trúc*** có chứa một thành phần dữ liệu là con trỏ trỏ tới một cấu trúc khác. Như vậy, mặc dù mỗi nút của danh sách được lưu trữ nằm rải rác ở bất kì đâu trong bộ nhớ, nhưng ta vẫn có thể truy nhập được nó thông qua địa chỉ được lưu trữ trong thành phần con trỏ của nút *đứng ngay trước nó*. Điều đó có nghĩa là, để truy nhập đến phần tử thứ *i* nào đó của danh sách móc nối ta cần phải biết địa chỉ của phần tử *i-1* đứng ngay trước nó, để truy nhập được đến phần tử *i-1* này ta cần phải biết địa chỉ của phần tử *i-2* đứng ngay trước phần tử *i-1*... cứ như vậy, ta thấy rằng để truy nhập đến một phần tử *i* bất kì nào đó của danh sách móc nối thì bao giờ ta cũng phải biết địa chỉ của *nút đầu tiên trong danh sách*. Hay nói cách khác, ta ***chỉ có thể truy nhập đến một phần tử nào đó trong danh sách một cách gián tiếp thông qua các phần tử đứng trước theo một chiều nhất định bắt đầu từ nút đầu tiên***. Do đó tốn thời gian truy nhập đến các phần tử trong danh sách, và thời gian này là không đồng đều giữa các nút khác nhau (*càng xa nút đầu tiên thì càng tốn thời gian hơn*). Danh sách loại này còn được gọi với một tên khác là ***danh sách nối đơn*** (*Singly linked list*).

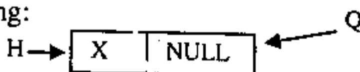
Việc quản lí danh sách móc nối thực chất quy về việc quản lí địa chỉ của nút đầu tiên thông qua con trỏ ***Dau***, con trỏ này phải ***không được thay đổi*** trong quá trình hoạt động của danh sách vì nó là *đầu mối duy nhất* để có thể truy nhập danh sách. Khi con trỏ ***Dau*** bằng *NULL* ta có một danh sách rỗng (*chưa có phần tử nào*). Để đánh dấu sự kết thúc của danh sách thì thành phần con trỏ của nút cuối cùng phải bằng *NULL* (*chưa trỏ đến nút nào*).

b) Các thao tác trên danh sách móc nối

Các thao tác chủ yếu trên cấu trúc dữ liệu kiểu danh sách nối đơn là các *phép bổ sung một phần tử vào danh sách*, *loại bỏ một phần tử ra khỏi danh sách*, *duyệt danh sách*, *ghép hai danh sách*... Để bổ sung một phần tử vào danh sách ta làm như sau: Giả sử rằng danh sách được quản lí bởi con trỏ đầu *H*, nút *X* cần bổ sung sẽ do con trỏ *Q* trỏ tới, vị trí cần bổ sung trong danh sách *H* do con trỏ *S* trỏ tới, khi đó ta cần xét các trường hợp sau đây:

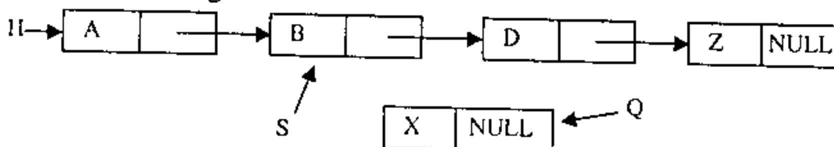
**** Trường hợp danh sách rỗng:***

- Trước khi bổ sung: *H=NULL* ;
- Sau khi bổ sung:

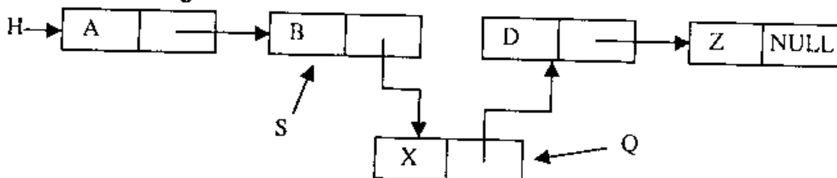


**** Trường hợp danh sách không rỗng***:

- Trước khi bổ sung:



- Sau khi bổ sung:

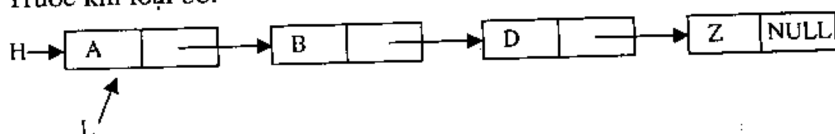


Hình 3.9. Các trường hợp bổ sung một phần tử vào danh sách nối đơn.

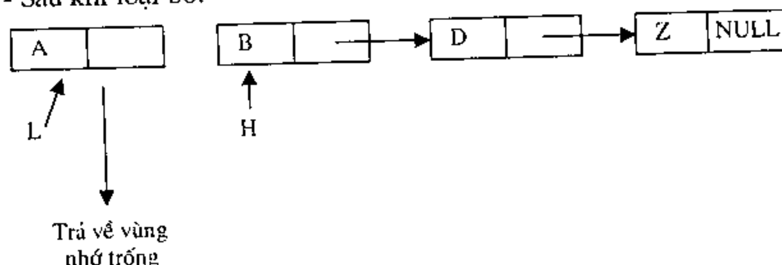
Để loại bỏ một phần tử do con trỏ *L* trỏ tới ra khỏi danh sách, ta cũng xét các trường hợp sau đây:

*** Trường hợp loại bỏ nút đầu danh sách**:

- Trước khi loại bỏ:

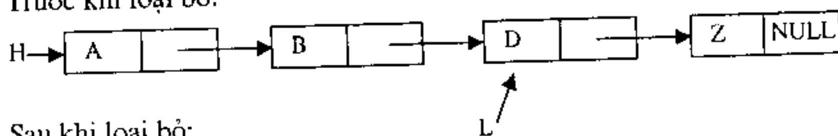


- Sau khi loại bỏ:

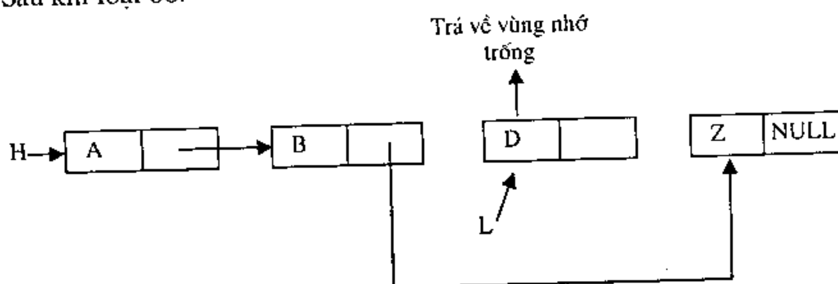


*** Trường hợp nút cần loại nằm sau nút đầu**:

- Trước khi loại bỏ:



- Sau khi loại bỏ:



Hình 3.10. Các trường hợp loại bỏ một phần tử ra khỏi danh sách nối đơn.

Để duyệt một danh sách nối đơn bao giờ ta cũng phải dùng một hoặc hai con trỏ (*không được dùng con trỏ đầu*) để di chuyển dọc theo danh sách. Còn phép ghép hai danh sách được thực hiện bằng cách ghép danh sách này vào cuối của danh sách kia (*lưu ý trường hợp danh sách rỗng*)... Các phép này sẽ được lần lượt giới thiệu qua các ví dụ dưới đây.

Ví dụ 3-28. Xét lại ***Ví dụ 3-24*** ở trên, ta thấy rằng việc không biết trước số lượng các bản tin cần lưu trữ là một hạn chế của việc dùng mảng. Nếu khai báo kích thước mảng quá lớn có thể gây ra lãng phí bộ nhớ. Ngược lại có thể bị thiếu bộ nhớ khi chạy chương trình. Một giải pháp tương đối hiệu quả trong trường hợp này là tổ chức dữ liệu dưới dạng danh sách móc nối.

```
/* ************************************************************************************************* */
#include "stdio.h"
#include "conio.h"
#include "alloc.h"
struct Date
{
     unsigned Ngay:5;
     unsigned Thang:4;
     unsigned Nam:15;
};
typedef struct pp
{
     struct Date ThoiGian;
     char DiaDiem[35];
     float LuongMua;
     int NhietDo:8;
     struct pp *Tiep;
} ThoiTiet; /* Định nghĩa một kiểu cấu trúc tự trỏ có tên là ThoiTiet */
/* ************************************************************************************************* */
int main()
{
     int NhietDoMax, xToaDo, yToaDo, SoPT=0;
     float LuongMuaTB;
      ThoiTiet  ThoiTietNgay, *Dau=NULL, *Duyet, *Q, *Cuoi;
     clrscr(); printf("\nNhap du lieu \n");
     do
     {     int TG;
           float Tam;
           printf("Ngay  ");
           xToaDo= wherex()+3; yToaDo= wherey();
           scanf("%d",&TG); ThoiTietNgay.ThoiGian.Ngay=TG;
           if(TG!=0)
           {
               gotoxy(xToaDo, yToaDo); xToaDo+=10;
               printf("Thang "); scanf("%d",&TG);
               ThoiTietNgay.ThoiGian.Thang=TG;
               gotoxy(xToaDo,yToaDo); printf("Nam "); scanf("%d", &TG);
               ThoiTietNgay.ThoiGian.Nam=TG;
               printf("Dia diem "); fflush(stdin);
               gets(ThoiTietNgay.DiaDiem);
               printf("Luong mua "); scanf("%f", &Tam);
               ThoiTietNgay.LuongMua=Tam;
               printf("Nhiet do "); scanf("%d", &TG);
               ThoiTietNgay.NhietDo=TG;
               printf("*****************\n");
           }
           if(ThoiTietNgay.ThoiGian.Ngay!=0)
           {
               /* Xin một nút để chứa một bản tin*/
               Q = (ThoiTiet*) malloc(sizeof (ThoiTiet));
```

```
            if(!Q) /* hết bộ nhớ */
            {
                printf("\nKhong du bo nho cho chuong trinh");
                exit(1); /* thoát khỏi chương trình một cách bình thường */
            }
            *Q=ThoiTietNgay; /* Đưa nội dung bản tin vừa nhập vào nút
            mới xin */
            /* Bắt đầu bổ sung nút mới xin vào danh sách */
            /* Trường hợp danh sách rỗng */
            if(!Dau)
            {
                Dau= Q; /* Nút đầu tiên trong danh sách*/
                Dau->Tiep=NULL; /*Phần tử cuối của danh sách*/
                Cuoi=Dau;/* Giữ lấy nút cuối cho lần nhập sau*/
            }
            else /* trường hợp không phải là nút đầu tiên */
            {
                Cuoi->Tiep = Q; /* Bổ sung nút mới vào cuối danh sách */
                Cuoi=Cuoi->Tiep; /* Di chuyển con trỏ Cuoi để trỏ vào
                phần tử cuối cùng trong danh sách */
                Cuoi->Tiep=NULL; /* Đánh dấu nút cuối */
            }
        }
}
while (ThoiTietNgay.ThoiGian.Ngay!=0); /* kết thúc nhập */
/* Tìm nhiệt độ cao nhất và lượng mưa trung bình – thao tác duyệt */
NhietDoMax= -128; LuongMuaTB=0;
Duyet= Dau; /* Bắt đầu xét từ phần tử đầu tiên trong danh sách */
if(Duyet==NULL) /* Danh sách rỗng */
{
    printf("\nChua co ban tin nao");
    return 0;
}
while(Duyet!=NULL) /* Bắt đầu duyệt */
{
      if(NhietDoMax < Duyet->NhietDo)
      {
          NhietDoMax = Duyet->NhietDo;
          Q= Duyet; /*Cho Q trỏ đến bản tin có nhiệt độ cao nhất */
      }
      LuongMuaTB += Duyet->LuongMua; /* Tính tổng lượng mưa */
      ++SoPT; /* Đếm số phần tử trong danh sách */
      Duyet=Duyet->Tiep; /* Di chuyển sang phần tử kế tiếp sau */
}
clrscr();
printf("\nNhiet do cao nhat quan sat duoc la: %3d", NhietDoMax);
printf("\nDo duoc tai %s", Q->DiaDiem);
printf("\nTrong ngay %d\nthang  %d\nnam  %d",
Q->ThoiGian.Ngay, Q->ThoiGian.Thang, Q->ThoiGian.Nam);
```

```
    printf("\nLuong mua trung binh cua khu vuc la: %10.2f",
    LuongMuaTB/SoPT);
    getch();
    return 0;
}
/* ******************************************************************************************* */
```

Bài tập:

- Trong ví dụ trên, nếu ta không dùng con trỏ *Cuoi* để luôn trỏ đến phần tử cuối cùng trong danh sách thì ta phải làm thế nào để đảm bảo chương trình vẫn hoạt động tốt? Viết lại chương trình trong trường hợp này.

- Nếu ta không sử dụng biến *SoPT* để đếm số phần tử có trong danh sách thì ta phải làm thế nào khi tính lượng mưa trung bình?

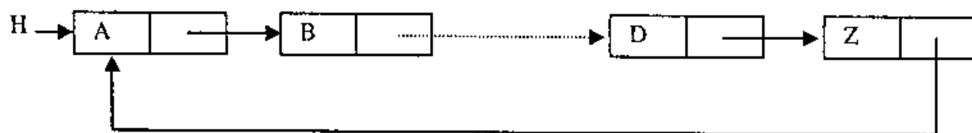
Trong chương 6 chúng ta sẽ đề cập nhiều hơn đến cách tổ chức dữ liệu dưới dạng danh sách móc nối cùng các biến thể của nó cũng như các thao tác cơ bản xử lí trên kiểu cấu trúc dữ liệu loại này (*các thao tác bổ sung, loại bỏ, tìm kiếm...*) trong việc giải quyết một số bài toán trong thực tế.

c) Một số biến thể của danh sách nối đơn

Một nhược điểm rất lớn của danh sách nối đơn là chỉ có phần tử đầu tiên của danh sách là được truy nhập trực tiếp còn các phần tử khác là gián tiếp qua các phần tử trước nó. Từ một phần tử ***i*** nào đó trong danh sách ta có thể dễ dàng truy nhập đến phần tử tiếp ngay sau nó, nhưng ta không có cách nào để có thể truy nhập đến phần tử đứng ngay trước nó được. Muốn làm được điều đó ta lại phải duyệt lại bắt đầu từ phần tử đầu tiên. Để giải quyết vấn đề này người ta đã đưa ra thêm một số biến thể của danh sách nối đơn để thuận tiện hơn trong khi viết chương trình:

**** Danh sách nối vòng***

Danh sách nối vòng là một danh sách nối đơn mà nút cuối cùng lại trỏ đến nút đầu tiên (*thành phần con trỏ của nút cuối cùng chứa địa chỉ của nút đầu tiên*).

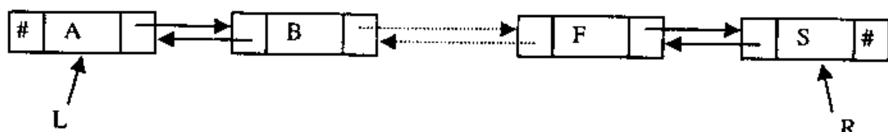


Hình 3.11. Hình ảnh của danh sách nối vòng trong bộ nhớ.

Với cải tiến này thì việc truy nhập vào danh sách móc nối sẽ linh hoạt hơn, từ bất kì một nút nào trong danh sách ta đều có thể truy nhập đến tất cả các nút khác. Mặc dầu vậy, trong quá trình xử lí ta nên chú ý đặt một nút nào đó một dấu hiệu đánh dấu sự kết thúc, nếu không có thể rơi vào vòng lặp vô hạn khi duyệt danh sách (*lí giải điều này được xem như bài tập*).

**** Danh sách nối kép***

Với các cấu trúc danh sách đã nêu ta chỉ có thể duyệt danh sách theo một chiều nhất định. Trong nhiều ứng dụng ta cần duyệt theo chiều ngược lại. Để có thể làm được điều đó mỗi nút trong danh sách cần được bổ sung thêm một trường con trỏ *trỏ tới nút kế trước nó* tạo thành một kiểu danh sách mới "*Danh sách nối kép*".



Hình 3.12. Hình ảnh của danh sách nối kép trong bộ nhớ.

Để thuận tiện trong quá trình truy nhập theo cả hai chiều ở đây ta sử dụng hai con trỏ *L* (*trỏ tới nút cực trái*) và *R* (*trỏ tới nút cực phải*). Khi danh sách rỗng thì *L* bằng *R* và bằng giá trị quy ước *NULL* (kí hiệu # ở đây quy ước là giá trị *NULL*).

3.2.5. Ngăn xếp (*Stack*)

1. Khái niệm

Ngăn xếp (*stack*) là một kiểu danh sách đặc biệt (*tuyến tính hoặc móc nối*) mà *phép bổ sung* và *phép loại bỏ* luôn luôn được thực hiện ở một đầu (*còn gọi là đỉnh*) của danh sách đó. Có thể hình dung cấu trúc dữ liệu này như cơ cấu hoạt động của băng đạn súng tiểu liên, thao tác lắp đạn vào hay lấy đạn ra (*bắn*) cũng chỉ thực hiện ở một đầu băng. Viên đạn nạp vào sau cùng sẽ nằm ở đỉnh *(được bắn trước*), còn viên đạn nạp vào đầu tiên sẽ nằm ở đáy băng (*được bắn sau cùng*). Chính vì nguyên tắc hoạt động theo kiểu " *Vào sau, ra trước*" này mà cấu trúc dữ liệu kiểu ngăn xếp còn được gọi một tên khác là cấu trúc LIFO (*Last-In-First-Out*).

2. Cấu trúc lưu trữ của ngăn xếp trong bộ nhớ

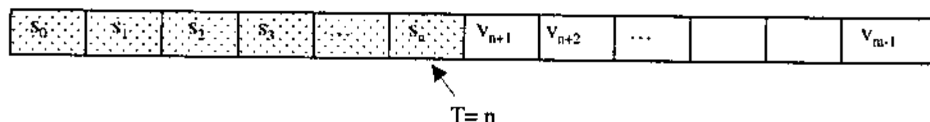
Nếu ngăn xếp *S* được lưu trữ kế tiếp trong bộ nhớ, người ta thường lưu trữ nó dưới dạng của một véc tơ V (*gồm m phần tử*) theo nguyên tắc sau:

- Phần tử thứ *i* của ngăn xếp sẽ được cất giữ tại phần tử thứ *i* của véc tơ.

- Đáy của ngăn xếp sẽ là phần tử đầu tiên của véc tơ V (*phần tử V[0]*).

- Đỉnh của ngăn xếp có thể là một biến trỏ *trỏ đến phần tử cuối cùng* trong danh sách (*có giá trị biến đổi khi chương trình hoạt động*) hoặc có thể là biến chỉ số của véc tơ. Để thuận tiện ta chọn đỉnh ngăn xếp sẽ là một biến *T* chứa *giá trị chỉ số* của phần tử thuộc véc tơ nhưng đang nằm ở đỉnh của ngăn xếp. Khi ngăn xếp rỗng *T* sẽ nhận một giá trị quy ước bằng -1.

- Các thao tác của ngăn xếp là lấy một phần tử ra khỏi ngăn xếp (*đọc phần tử V[T] để xử lí, giảm T đi 1*) và bổ sung một phần tử *X* vào trong ngăn xếp (*Tăng T lên 1, đưa giá trị của phần tử X vào V[T]*).

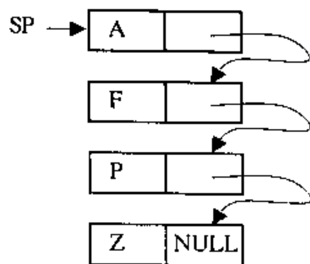


Hình 3.13. Hình ảnh của cấu trúc dữ liệu ngăn xếp được lưu trữ kế tiếp trong bộ nhớ.

Nếu ngăn xếp được lưu trữ móc nối với nhau theo dạng danh sách nối đơn thì nó sẽ có dạng như *hình 3.14*.

Trong trường hợp này ta có thể coi đỉnh của ngăn xếp chính là nút đầu tiên trong danh sách. Phép bổ sung và loại bỏ luôn luôn được thực hiện tại nút đầu tiên

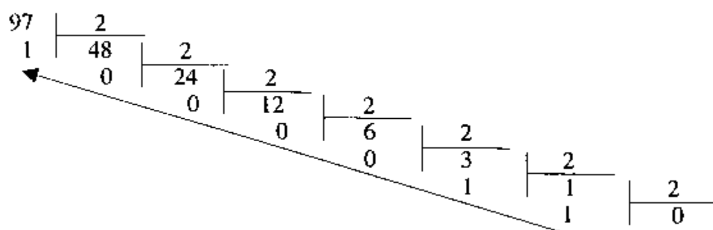
này. Đáy ngăn xếp được đánh dấu bằng nút cuối cùng trong danh sách (*có thành phần con trỏ NULL*). Như vậy ngăn xếp sẽ rỗng khi con trỏ đầu *SP* (*đỉnh ngăn xếp*) trỏ đến phần tử có thành phần con trỏ có giá trị *NULL*.



Hình 3.14. Ngăn xếp được tổ chức móc nối.

Cấu trúc dữ liệu kiểu ngăn xếp có nhiều ứng dụng trong các bài toán thực tiễn và trong ngành khoa học máy tính. Người ta có thể dùng cấu trúc dữ liệu kiểu ngăn xếp để cài đặt cho các giải thuật đệ quy, cài đặt các hàm trong lập trình cấu trúc, đổi cơ số, biểu diễn các biểu thức toán học theo kí pháp *Ba Lan* trong máy tính...

Ví dụ 3-29. Hãy viết chương trình thực hiện đổi một số nguyên ở dạng hệ cơ số *mười* sang hệ cơ số *hai* (*số nhị phân*). Trước khi giải được bài toán, cần phải hiểu rõ thuật toán chuyển đổi từ hệ cơ số *10* sang hệ cơ số 2. Ví dụ để chuyển số nguyên 97 sang hệ nhị phân (*cơ số 2, chỉ gồm toàn số 0 và 1*) ta làm như sau:



Ta thực hiện phép chia liên tiếp cho 2, cho đến khi thương số thu được bằng không. Các số dư thu được nếu viết theo thứ tự ngược lại chính là số *97* biểu diễn trong hệ nhị phân (*số 1100001*). Như vậy, mỗi khi thực hiện xong một phép chia, ta sẽ lưu số dư vào trong ngăn xếp cho đến khi kết thúc. Thứ tự các số dư này lấy dần ra sẽ là số nhị phân cần tìm, ngăn xếp được tổ chức móc nối trong bộ nhớ.

```
/* ******************************************************************************** */
#include "stdio.h"
#include "conio.h"
#include "alloc.h"
#include "stdlib.h"
typedef struct NP
{
    unsigned char Bit;
    struct NP *Tiep;
} NhiPhan; /* Định nghĩa kiểu dữ liệu nhị phân */
NhiPhan *Dinh= NULL; /* Đỉnh ngăn xếp */
/* ******************************************************************************** */
int main()
{
    NhiPhan *P; /* con trỏ dùng để xin nút mới */
    int n;
```

```
    unsigned char Du;
    printf("\nNhap vao so nguyen");
    scanf("%d", &n);
    /* Bắt đầu chia */
    do
    {
        Du= n%2;
        n /= 2;
        if((P= (NhiPhan*)(malloc(sizeof(NhiPhan))))==NULL)
        {
            printf("\nKhong du bo nho");
            exit(-1);
        }
        else
        {
            P->Bit=Du;
            /* Đưa phần tử số dư vừa tính vào ngăn xếp */
            P->Tiep= Dinh;
            Dinh= P; /* Chỉnh con trỏ Dinh trỏ vào đỉnh ngăn xếp */
        }
    }
    while(n!=0); /* Chỉ kết thúc khi số bị chia đã bằng không */
    /*Đưa kết quả ra màn hình*/
    printf("So nguyen da cho co bieu dien trong he co so hai la: ");
    /* Lấy từng phần tử ra khỏi ngăn xếp */
    while(Dinh!=NULL) /* Còn chưa gặp đáy ngăn xếp */
    {
        printf("%1d", Dinh->Bit); /* Hiện từng bít */
        P=Dinh;
        Dinh=Dinh->Tiep;
        free(P); /* Hiện xong, giải phóng vùng nhớ */
    }
    getch();
    return 0;
}
/* ************************************************************************************** */
```

Bài tập:

Viết lại chương trình trên nhưng tổ chức ngăn xếp dưới dạng danh sách tuyến tính.

3.2.6. Hàng đợi (*Queue*)

1. Khái niệm

Hàng đợi là một dạng đặc biệt của danh sách (*tuyến tính hoặc móc nối*) mà phép bổ sung được thực hiện ở một đầu (*gọi là lối sau*), còn phép loại bỏ được thực hiện ở một đầu khác (*gọi là lối trước*). Việc xếp hàng để mua vé xem phim cho ta một hình ảnh rõ nét của hàng đợi. Những người đến trước (*xếp hàng trước*) thì được mua trước (*ra về trước - loại bỏ ra khỏi hàng trước*), những người đến sau thì phải mua sau (*ra khỏi hàng sau*). Chính vì hàng đợi hoạt động theo nguyên tắc "*Vào*

trước ra trước" này mà nó còn có một tên khác là cấu trúc dữ liệu kiểu *FIFO* (*First-In-First-Out*).

2. Cấu trúc lưu trữ của hàng đợi trong bộ nhớ

Nếu hàng đợi ***Q*** được lưu trữ kế tiếp trong bộ nhớ, người ta thường lưu trữ nó dưới dạng của một véc tơ *V* (*kích thước m*) theo nguyên tắc sau:

- Mỗi phần tử của hàng đợi sẽ được lưu trữ trong một phần tử của véc tơ *V*.

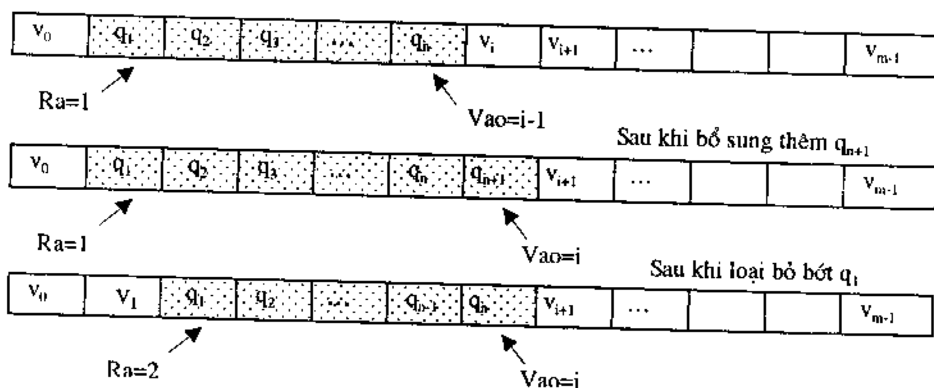
- Lối vào của hàng đợi được đánh dấu bằng một biến ***Vao*** lưu giữ *chỉ số* của phần tử trong *V* tương ứng với phần tử cuối cùng của hàng đợi.

- Lối ra của hàng đợi được đánh dấu bằng một biến ***Ra*** lưu giữ chỉ số của phần tử trong *V* tương ứng với phần tử đầu tiên của hàng đợi.

- Khi biến ***Vao < Ra*** thì hàng đợi rỗng.

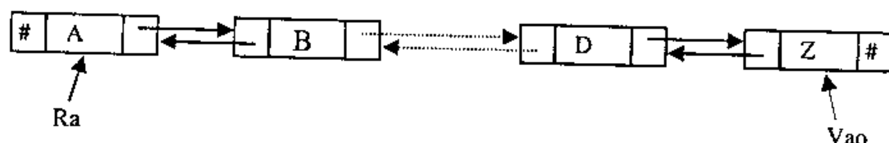
- Thao tác bổ sung (*đưa vào lối vào*) trên hàng đợi thực chất là tiến hành tăng biến ***Vao*** lên 1 (*mở rộng hàng đợi về bên phải của V*) rồi đưa nội dung của phần tử cần bổ sung vào vị trí *V[Vao]*.

- Thao tác loại bỏ (*đưa ra khỏi lối ra*) trên hàng đợi thực chất là tiến hành lấy nội dung của phần tử *V[Ra]* để xử lí, sau đó tăng biến ***Ra*** lên 1 (*thu hẹp hàng đợi về bên phải của V*).



Hình 3.15. Hoạt động của cấu trúc dữ liệu hàng đợi lưu trữ kế tiếp trong bộ nhớ tại các thời điểm.

Nếu hàng đợi được lưu trữ dưới dạng danh sách móc nối thì nó thường được lưu trữ dưới dạng danh sách nối kép và được mô tả như sau:



Hình 3.16. Hình ảnh của hàng đợi được tổ chức dưới dạng danh sách nối kép trong bộ nhớ.

Phép bổ sung sẽ được thực hiện tại con trỏ ***Vao***, phép loại bỏ sẽ được thực hiện tại con trỏ ***Ra***. Khi *Vao = Ra = NULL* thì hàng đợi rỗng.

Ví dụ 3-30. Viết chương trình ghi lại tên khách hàng đã đăng kí đặt vé tầu. In ra danh sách khách hàng theo thứ tự: người đăng kí trước được in trước, người đăng kí sau được in sau, dữ liệu được tổ chức dưới dạng danh sách nối đơn.

```
/* ********************************************************************************** */
#include "stdio.h"
#include "conio.h"
#include "alloc.h"
#include "stdlib.h"
#include "string.h"
typedef struct KH
{
    char HoTen[30];
    struct KH *Truoc;
} KhachHang;
KhachHang *Ra= NULL, *Vao= NULL;
/* ********************************************************************************** */
int main()
{
    KhachHang *kh,*p, *q;
    char Ten[30];
    int n=0; /* Ban đầu chưa có khách hàng nào */
    /* Nhập vào danh sách khách hàng cho đến khi gặp một tên rỗng */
    do
    {
        printf("\nNhap vao thong tin cua khach hang %d \n", n+1);
        fflush(stdin);
        printf("Ho ten  "); gets(Ten);
        if(strcmp(Ten, "")!=0)
        {
            if((kh=(KhachHang*)(malloc(sizeof(KhachHang))))==NULL)
            {
                printf("\nKhong du bo nho");
                exit(-1);
            }
            else
            {
                n++;
                strcpy(kh->HoTen, Ten);
                /* Đưa khách hàng vừa nhập vào hàng đợi */
                if(Ra==NULL)
                {
                    Ra=Vao=kh;
                    Vao->Truoc=NULL;
                }
                else
                {
                    kh->Truoc= Vao; /* Đưa vào lối vào */
                    Vao= kh; /* Chỉnh lại con trỏ Vao trỏ đến phần tử mới */
                }
            }
```

```
        }
    }
    while(strcmp(Ten, "")!=0); /* Kết thúc nhập khi gặp tên rỗng */
    /* Lấy các phần tử ra khỏi hàng đợi  và in danh sách khách hàng*/
    printf("\nDanh sach khach  hang da dang ky\n");
    n= 0;
    while(Ra!=NULL)
    {
        p= Vao;
        /* Di chuyển p đến phần tử sát lối ra */
        while(p!=Ra)
        {
            q=p;
            p=q->Truoc;
        }
        printf("%d. %20s\n",++n, p->HoTen);
        if(Ra==Vao)
        {
            free(Ra);
            Ra=Vao=NULL;
        }
        else /* Loại bỏ một phần tử ở lối ra của hàng đợi */
        {
            free(Ra);
            q->Truoc=NULL;
            Ra=q;
        }
    }
    getch();
    return 0;
}
/* ************************************************************************************************ */
```

Bài tập:

Viết lại chương trình trên nhưng tổ chức dữ liệu theo kiểu *danh sách nối kép* và *danh sách tuyến tính.*

3.3. CÂU HỎI VÀ BÀI TẬP

1. Dựa trên những đặc điểm gì để phân biệt véc tơ và danh sách tuyến tính ? Ta có thể dùng mảng để lưu trữ cho danh sách tuyến tính được không? Tại sao?

2*. Cho hệ phương trình đại số tuyến tính có dạng:

$a_{11}x_1=b_1$

$a_{21}x_1+ a_{22}x_2= b_2$

...

$a_{n1}x_1+ a_{n2}x_2+\ldots+ a_{nn}x_n= b_n$

Để tiết kiệm, người ta lưu trữ ma trận hệ số của hệ phương trình dưới dạng một véc tơ. Như vậy chỉ cần $n(n+1)/2$ phần tử thay vì phải dùng n^2 phần tử. Hãy viết chương trình giải hệ phương trình ứng với cấu trúc dữ liệu đã chọn.

3. Hãy nhập một dãy *n* điểm, mỗi điểm có tọa độ thực. Xét xem có bao nhiêu điểm nằm bên trong góc phần tư thứ *2*, bao nhiêu điểm nằm bên trong góc phần tư thứ *3*.

4. Nhập ma trận cấp *mxn*. Tìm một phần tử *Max* của mỗi hàng. In mỗi phần tử tìm được trên một dòng cùng với vị trí của phần tử đó trong ma trận.

5. Phân biệt cấu trúc với hợp? Cách sử dụng của từng loại? Con trỏ là gì? Có những loại con trỏ nào? Sử dụng con trỏ trong ngôn ngữ lập trình C như thế nào?

6. Viết chương trình nhập từ bàn phím một danh sách sinh viên gồm: Họ và tên, năm sinh, giới tính, quê quán. Đưa ra màn hình danh sách này và danh sách các nữ sinh viên sinh sau năm *1980*.

7. Viết chương trình thực hiện các việc sau:

a) Nhập từ bàn phím một danh sách bản ghi gồm các trường:

TenThuoc (độ dài tối đa là 30) chứa tên thuốc

NamHetHan chứa năm hết hạn của thuốc đó

b) Xóa khỏi danh sách những loại thuốc có năm hết hạn trước năm 2004.

c) Đưa ra màn hình tên các thuốc đến năm 2004 là hết hạn.

8. Cho một danh sách hàng hóa gồm tên mặt hàng, số lượng, đơn giá, thành tiền (số *lượng nhân với đơn giá*). Viết chương trình thực hiện các công việc sau:

a) Nhập dữ liệu từ bàn phím, kết thúc nhập khi tên mặt hàng bằng ''.*

b) Đưa ra màn hình danh sách đã nhập.

c) Thêm vào cuối danh sách một mặt hàng mới, đưa kết quả ra màn hình.

d) Đưa ra màn hình tên và số lượng của các mặt hàng có số lượng nhỏ hơn 5.

9. Cho một danh sách thành tích thi đấu bóng đá của *18* đội tuyển gồm: Tên đội bóng, số bàn thắng, số bàn thua, số thẻ vàng, số thẻ đỏ. Viết chương trình thực hiện các việc sau:

a) Nhập dữ liệu từ bàn phím.

b) Đọc vào tên một đội bóng, đưa ra màn hình thành tích của đội này. Nếu tên đội bóng không có trong danh sách thì thông báo không có.

c) Mỗi bàn thắng được thưởng 10 điểm, mỗi bàn thua bị phạt 5 điểm, mỗi thẻ vàng bị phạt 2 điểm, mỗi thẻ đỏ bị phạt 5 điểm. Tính và đưa ra màn hình số điểm của các đội.

10. Viết lại ví dụ 3-30 bằng cách tổ chức dữ liệu theo kiểu danh sách nối vòng và danh sách nối kép.

11. Ta có thể lưu trữ hai ngăn xếp trong một véc tơ được không ? Nếu câu trả lời là có thì phải làm như thế nào cho hiệu quả ?

12. Việc lưu trữ hàng đợi dưới dạng kế tiếp (*thông qua véc tơ*) có một nhược điểm rất lớn là hàng đợi sẽ bị dịch chuyển dần dần về phía lối vào cho đến khi không thể dịch được nữa (*không thể bổ sung được nữa*) mặc dầu vùng nhớ dành cho nó vẫn còn khá nhiều ở lối ra. Ta có thể có cách tổ chức hàng đợi nào để giải quyết vấn đề trên hay không ? Nếu có thì làm như thế nào cho hiệu quả ?

13. Nếu lưu trữ hàng đợi theo kiểu danh sách nối đơn thì ta cần phải làm gì với các phép bổ sung và loại bỏ ?

14. Hãy viết chương trình cộng hai đa thức. Mỗi đa thức được lưu trữ dưới dạng một bộ giá trị như sau: (*Số phần tử của đa thức, các số mũ và hệ số khác không*). Ví dụ đa thức $x^{1000}+1$ được biểu diễn (*2, 1000, 1, 0, 1*) hoặc đa thức $x^7 + 3x^4 - 5x + 3$ được biểu diễn là (*4, 7, 1, 4, 3, 1, -5, 0, 3*).

15. Nhập một danh sách *n* học sinh với các thuộc tính: họ tên, năm sinh và tổng điểm. Sắp xếp danh sách theo thứ tự giảm của tổng điểm. Khi tổng điểm bằng nhau thì học sinh có năm sinh nhỏ hơn được xếp trước. In danh sách học sinh đã sắp xếp ra màn hình sao cho tất cả các chữ cái của họ tên chuyển thành chữ hoa.

16. Cho một danh sách liên kết gồm các cấu trúc kiểu *TS* được định nghĩa như sau:

```
struct TS
{
        char HT[26];
        float TongDiem;
        struct TS *Tiep;
};
```

Cho biết con trỏ *Dau* trỏ tới đầu danh sách. Viết chương trình thực hiện các công việc sau:

a) Bổ sung một thí sinh vào cuối danh sách.

b) Lập một danh sách mới gồm các thí sinh trúng tuyển (*điểm chuẩn nhập từ bàn phím*).

17. Cho danh sách liên kết gồm các cấu trúc có thành phần như sau: Họ và tên, năm sinh. Viết chương trình thực hiện các việc sau:

a) In các học sinh có năm sinh từ 1972 trở lại đây.

b) Xóa khỏi danh sách các học sinh sinh năm 1974.

18*. Cho một văn bản không quá 60 dòng, mỗi dòng không quá 80 kí tự. Viết chương trình tính số lần xuất hiện của từng chữ cái trong văn bản đó và tần suất xuất hiện của chúng. Đưa ra màn hình.

19. Viết chương trình đọc vào một xâu. Không dùng xâu trung gian, hãy đảo các kí tự trong xâu theo thứ tự ngược lại. Đưa xâu ban đầu và xâu đã đảo ra màn hình.

20. Viết chương trình đảo các số trong một số nguyên cho trước.

Chương 4
HÀM VÀ TỔ CHỨC CHƯƠNG TRÌNH VỀ MẶT CẤU TRÚC

MỤC TIÊU CỦA CHƯƠNG NÀY

- *Biết cách phân chia chương trình thành các mô đun*
- *Nắm vững các cách truyền thông tin giữa các hàm*
- *Biết cách kết hợp nhuần nhuyễn các kiểu cấu trúc dữ liệu cơ bản để xây dựng các ứng dụng thực tế.*

4.1. PHƯƠNG PHÁP TỔ CHỨC CHƯƠNG TRÌNH THEO MÔ ĐUN

Nguyên lí chủ đạo của kĩ thuật lập trình có cấu trúc là sử dụng khái niệm trừu tượng hóa chức năng để phân rã bài toán thành nhiều bài toán nhỏ hơn, mỗi bài toán con lại trở thành một đơn vị chương trình tương đối độc lập (*có cấu trúc riêng, có biến riêng...*). Trong quá trình làm việc ta chỉ quan tâm liệu một hàm có thể làm được công việc cụ thể gì (*yêu cầu đầu vào và kết quả đầu ra*) mà không cần quan tâm hàm đó phải làm như thế nào để đạt được kết quả ấy. Việc thiết kế chương trình như vậy cho phép chương trình được tổ chức một cách sáng sủa hơn, dễ bảo dưỡng hơn và dễ mang đi hơn.

Có nhiều cách khác nhau để phân mảnh chương trình thành các mô đun nhỏ hơn, một cách thường được sử dụng trong khi viết các chương trình ứng dụng đó là phương pháp thiết kế ***Trên xuống*** (*Top-Down*). Tư tưởng của phương pháp như sau: *Tiến hành phân tích bài toán bằng cách đi dần từ một mô tả đại thể đến những mô tả chi tiết thông qua nhiều mức*. Sự chuyển dịch từ một mức tới mức tiếp theo thực chất là sự phân rã mỗi chức năng ở mức trên thành một số chức năng con ở mức dưới mà kết quả là ta thu được một cây phân cấp của bài toán ban đầu.

Ví dụ 4-1: Hãy viết chương trình quản lí và bảo trì các hồ sơ về học bổng của các sinh viên trong diện được tài trợ, đồng thời phải thường kì lập báo cáo lên cấp có thẩm quyền.

Trước hết ta phải xác định được bài toán như sau:

- ***Đầu vào:*** *Tập hồ sơ sinh viên bao gồm các bản ghi về các thông tin liên quan đến học bổng của sinh viên.*

- ***Đầu ra***: Phải giải quyết được các yêu cầu sau đây:

a) Tìm lại và hiển thị được bản ghi của bất kì sinh viên nào khi có yêu cầu.

b) Có thể cập nhật được một bản ghi của một sinh viên cho trước.

c) In báo cáo.

Để làm được điều đó chương trình cần có ba nhiệm vụ chính như sau:

a) ***Đọc tệp*** *để lấy thông tin.*

b) ***Xử lí*** *tệp.*

c) ***Ghi tệp.***

Các nhiệm vụ này lại được chia ra làm các nhiệm vụ nhỏ hơn, ví dụ nhiệm vụ xử lí tệp lại bao gồm ba nhiệm vụ nhỏ cần giải quyết như sau:

b.1. Tìm lại bản ghi của một sinh viên cho trước.

b.1.1. Tìm kiếm

b.1.2. Hiển thị bản ghi

b.2. Cập nhật thông tin trong bản ghi sinh viên

b.2.1. Tìm kiếm

b.2.2. Sửa đổi

b.3. In bản tổng kết những thông tin về các sinh viên được học bổng.

...

Cứ như vậy ta sẽ được một cấu trúc phân cấp dạng hình cây cho bài toán ban đầu với mỗi một nút lá (*nút không có nút con nào*) sẽ là một hàm. Sự kết hợp các hàm này trong một chương trình chính là lời giải cho bài toán đã cho. Trong các phần sau ta sẽ nghiên cứu nguyên tắc xây dựng và sử dụng các hàm cũng như các cách trao đổi thông tin (*truyền tham số*) giữa các hàm trong một chương trình C.

4.2. CẤU TRÚC TỔNG QUÁT CỦA MỘT CHƯƠNG TRÌNH C

Một chương trình viết theo ngôn ngữ lập trình C là một dãy các hàm (*mỗi hàm sẽ thực hiện một phần việc nào đó*) để giải quyết một công việc trọn vẹn, trong đó phải có một hàm chính gọi là hàm ***main***. Thứ tự của các hàm trong chương trình có thể tùy ý (*tuy nhiên chúng phải được khai báo trước khi sử dụng*), nhưng chương trình bao giờ cũng *chỉ được thực hiện bắt đầu từ hàm* ***main***. Có nghĩa là chương trình sẽ chỉ thực hiện bắt đầu từ câu lệnh sau dấu '*{*' của thân hàm ***main*** cho đến khi gặp dấu '*}*' đánh dấu sự kết thúc của hàm ***main***. Các hàm khác sẽ chỉ được thực hiện qua các ***lời gọi hàm*** nằm bên trong thân của hàm ***main*** mà thôi.

Cấu trúc tổng quát của một chương trình C sẽ có dạng như sau:

+ *Các chỉ thị tiền xử lí:*

#include

#define

+ *Các định nghĩa, khai báo của các kiểu dữ liệu và các biến ngoài.*

+ *Khai báo nguyên mẫu của các hàm.*

+ *Hàm* ***main***

+ *Định nghĩa của các hàm.*

Chú ý:

- Các chỉ thị tiền xử lí có thể nằm ở bất cứ đâu trong chương trình và nó sẽ có hiệu lực từ khi xuất hiện.

- Các định nghĩa và khai báo các kiểu dữ liệu và biến ngoài có thể nằm xen kẽ giữa các hàm nhưng sẽ không thể được sử dụng trong các hàm được định nghĩa trước khi định nghĩa biến hay kiểu dữ liệu đó.

- Vị trí của hàm ***main*** cũng có thể nằm xen kẽ giữa các hàm khác.

- Các hàm không thể khai báo lồng nhau, nghĩa là không thể khai báo một hàm nằm trong một hàm khác.

- Một hàm không nhất thiết phải khai báo nguyên mẫu nhưng nên có vì nó cho phép chương trình biên dịch phát hiện lỗi khi gọi hàm (*do đối số không đúng*) hoặc tự động chuyển đổi kiểu cho phù hợp với lời gọi hàm.

4.3. QUY TẮC XÂY DỰNG VÀ SỬ DỤNG MỘT HÀM

4.3.1. Quy tắc xây dựng một hàm

- Mỗi hàm phải có một tên theo quy tắc đặt tên đã trình bày trong chương 1. Trong một chương trình không được phép có hai hàm trùng tên nhau.

- Mỗi hàm thường có các giá trị ***Đầu vào*** và các giá trị ***Đầu ra***. Các giá trị đầu vào được truyền thông qua *danh sách tham số* của hàm hoặc thông qua các *biến toàn cục*, còn các giá trị đầu ra được gửi trả về ***nơi gọi nó*** thông qua câu lệnh ***return (Biểu thức)*** khi hàm kết thúc, qua địa chỉ của biến hoặc qua một biến toàn cục. Khi một hàm không có đối số (*hoặc không có giá trị trả về*) sẽ được khai báo đối số (*hoặc giá trị trả về*) dạng không kiểu ***void***. Trong trường hợp này hàm sẽ có tác dụng giống như thủ tục (*procedure*) trong *Pascal*.

- Các hàm có vai trò ngang nhau trong chương trình.

- Mỗi hàm trong ngôn ngữ lập trình C, về nguyên tắc bao gồm hai phần, một phần gọi là ***Nguyên mẫu*** của hàm (*Prototype*) được khai báo trước khi hàm được sử dụng và phần còn lại gọi là ***Phần định nghĩa*** của hàm.

Phần nguyên mẫu của hàm sẽ mô tả đầy đủ các thông tin cần thiết liên quan đến một hàm, đó là tên hàm, đầu vào (*danh sách tham số*) và đầu ra (*giá trị trả về*) của hàm theo mẫu sau:

KiểuGiáTrịTrảVề   TênHàm(DanhSáchThamSố);

Trong đó, danh sách tham số được phân tách nhau bởi dấu phẩy ',' có thể chỉ liệt kê các kiểu tương ứng với các tham số mà thôi (*không cần tên biến*).

Ví dụ 4-2. Hàm dùng để tìm số lớn nhất trong ba số có danh sách tham số tương ứng (*giá trị đầu vào*) là ba biến thực và giá trị trả về (*đầu ra*) là số lớn nhất trong ba số đã cho được mô tả nguyên mẫu như sau:

float   TimMax (float, float, float);

Giá trị trả về   Tên hàm   Danh sách tham số

Phần định nghĩa của hàm lại bao gồm hai bộ phận đó là ***Dòng tiêu đề*** và ***Thân hàm***. Dòng tiêu đề thực chất là nguyên mẫu của hàm được viết lại nhưng phải có tên các tham số tương ứng với các kiểu dữ liệu (*phân tách nhau bởi dấu phẩy*). Các tham số này được gọi là các *tham số hình thức*. ***Phần thân hàm*** thực chất là một ***Khối lệnh*** nhằm thực hiện nhiệm vụ của hàm với nguyên liệu ban đầu là giá trị các tham số và kết thúc bằng một lệnh ***return(BiểuThức);*** để trả kết quả tìm được cho nơi gọi nó. Câu lệnh này có thể vắng mặt nếu hàm không có giá trị trả về (*trả về kiểu void*). Trong thân hàm có thể có nhiều hơn một câu lệnh *return* ở những chỗ khác nhau, khi gặp câu lệnh này máy sẽ tính giá trị ***BiểuThức*** (*nếu có*) đặt sau nó, xóa các biến cục bộ, xóa các tham số, gán giá trị của biểu thức tính được cho hàm rồi thoát khỏi hàm, trả quyền điều khiển về cho hàm gọi nó để thực hiện lệnh tiếp theo ngay sau lời gọi hàm.

Sau khi định nghĩa, để có thể sử dụng được hàm ta phải thực hiện một ***lời gọi hàm*** theo mẫu sau trong thân của hàm ***main*** hoặc trong một hàm khác được gọi bởi hàm ***main***:

TênHàm (Danh sách các tham số thực);

Với điều kiện *số* tham số thực phải bằng *số* tham số hình thức và kiểu của các tham số thực phải phù hợp với kiểu của các tham số hình thức tương ứng.

Ví dụ 4-3. Ta có thể viết cụ thể ví dụ ở 4-2 như sau:

```
/* ************************************************************************************ */
#include "stdio.h"
#include "conio.h"
float TimMax(float,float,float); /* Nguyên mẫu của hàm */
/* ************************************************************************************ */
int main()
{
      float x, y, z; /*Các tham số thực của hàm*/
      clrscr();
      printf("\nHay nhap ba so can tim Max x, y, z= ");
      scanf("%f%f%f", &x, &y, &z);
      printf("\nGia tri lon nhat trong ba so x= %10.2f , y= %10.2f, z= %10.2f\
      la Max= %10.2f", x, y, z, TimMax(x, y, z) (24));
      getch();
      return 0;
} /* Kết thúc hàm main */
/* ************************************************************************************ */
/* Định nghĩa của hàm TimMax */
float TimMax(float i, float j, float k) (25)  /*Dòng tiêu đề*/
{ /* Bắt đầu phần thân hàm TimMax */
      float Max; /* Biến cục bộ của hàm */
      Max= i > j ? i : j;  /* Tìm số lớn nhất trong hai số i và j */
      return (Max > k ? Max : k); /*So sánh với số còn lại*/
} /* Kết thúc hàm TimMax*/
```

[24] Lời gọi hàm TimMax bên trong hàm main.

[25] Các *tham số hình thức* có thể cùng tên hoặc khác tên với các *tham số thực sự* trong lời gọi của hàm. Chúng ta sẽ hiểu rõ hơn trong mục 4.3.2

4.3.2. Hoạt động của hàm

Trong hàm *main*, khi gặp một lời gọi hàm (*như trong câu lệnh printf() của ví dụ trên*), máy sẽ chuyển đến thực hiện hàm được gọi với các giá trị ban đầu (*đầu vào của hàm*) là bộ giá trị được truyền cho các tham số theo trình tự sau đây:

- Máy cấp phát bộ nhớ cho các tham số hình thức và các biến cục bộ.
- Gán giá trị của các *tham số thực* trong lời gọi hàm cho các tham số hình thức tương ứng vừa cấp phát bộ nhớ (*làm việc trên bản Copy của các tham số thực*).

- Thực hiện các câu lệnh trong thân hàm (*chỉ có thể thao tác thực sự trên các biến cục bộ và các biến toàn cục, còn bản thân các tham số thực không hề bị ảnh hưởng gì*). Ta có thể hiểu rõ điều này qua ví dụ sau:

Ví dụ 4-4. Viết hàm thực hiện việc hoán đổi giá trị của hai biến nào đó.

```
/* ************************************************************************************************ */
#include "stdio.h"
#include "conio.h"
void  HoanVi(float, float); /* Nguyên mẫu của hàm */
/* ************************************************************************************************ */
int main()
{
      float x, y; /*Các tham số thực của hàm*/
      clrscr();
      printf("\nHay nhap hai  so can hoan vi x, y= ");
      scanf("%f%f", &x, &y);
      printf("\nCac gia tri truoc khi hoan vi la x= %10.2f va y=10.2f", x, y);
      HoanVi(x, y); /*Gọi hàm hoán vị với tham số thực là x và y */
      printf("\nCac gia tri sau khi hoan vi la x= %10.2f va y=10.2f" , x, y);
      getch();
      return 0;
}/* Kết thúc hàm main */
/* ************************************************************************************************ */
/* Định nghĩa của hàm HoanVi */
void HoanVi(float i, float j)  /*Dòng tiêu đề với các tham số hình thức */
{/* Bắt đầu phần thân hàm HoanVi */
      float Tam; /*Biến cục bộ của hàm*/
      Tam = i;
      i = j;
      j = Tam;
      return;
} /* Kết thúc hàm HoanVi */
/* ************************************************************************************************ */
```

Khi thực hiện chương trình trên với các giá trị *x =10, y =35* thì kết quả sau khi gọi hàm *HoanVi* vẫn là *x =10, y =35*. Điều này có vẻ dường như hàm *HoanVi* đã không thực hiện gì. Trong thực tế thì hàm *HoanVi* đã làm việc, tuy nhiên việc tráo đổi này chỉ diễn ra trên bản Copy của các tham số thực mà thôi, bản thân các biến này không hề bị ảnh hưởng bởi lời gọi hàm. Ra khỏi hàm, các giá trị đã hoán đổi trên các bản Copy cũng mất theo và do đó ta có cảm giác chương trình không làm gì cả. Việc truyền tham số cho hàm theo kiểu này người ta gọi là *truyền theo tham trị*.

Để có thể xử lí tình huống này, thay vì sẽ làm việc trên các bản Copy của tham số thực ta yêu cầu hàm làm việc trực tiếp trên các biến đó bằng cách gửi địa chỉ của các tham số thực cho danh sách các tham số hình thức tương ứng. Cách truyền tham số cho hàm theo cách này gọi là *truyền theo địa chỉ*. Trong ngôn ngữ lập trình C, để truyền tham số theo địa chỉ thì các *tham số hình thức* trong định nghĩa của hàm phải là các con trỏ, còn danh sách các *tham số thực* trong lời gọi hàm phải là *danh sách các địa chỉ* của các biến đó. Ta có thể viết lại ví dụ trên như sau:

Ví dụ 4-5. Viết hàm thực hiện việc hoán đổi giá trị của hai biến nào đó (*bản hàm thực hiện truyền tham số theo địa chỉ*).

```
/* ************************************************************************************** */
#include "stdio.h"
#include "conio.h"
void  HoanVi(float *, float *); /* Nguyên mẫu của hàm */
/* ************************************************************************************** */
int main()
{
        float x, y; /*Các tham số thực của hàm*/
        clrscr();
        printf("\nHay nhap hai  so can hoan vi x, y= ");
        scanf("%f%f", &x, &y);
        printf("\nCac gia tri truoc khi hoan vi la x= %10.2f va y=10.2f" , x, y);
        HoanVi(&x, &y); /*Gọi hàm hoán vị với tham số thực là địa chỉ */
        printf("\nCac gia tri sau khi hoan vi la x= %10.2f va y=10.2f" , x, y);
        getch();
        return 0;
}/* Kết thúc hàm main */
/* ************************************************************************************** */
/* Định nghĩa của hàm HoanVi */
void HoanVi(float* i, float* j)  /*Dòng tiêu đề*/
{/*Bắt đầu phần thân hàm HoanVi */
        float Tam; /*Biến cục bộ của hàm*/
        Tam= *i;
        *i = *j;
        *j = Tam;
        return;
} /* Kết thúc hàm HoanVi */
/* ************************************************************************************** */
```

- Khi gặp câu lệnh ***return*** hoặc dấu '*}*' cuối cùng của thân hàm thì máy sẽ xóa các tham số, các biến cục bộ (*do đó các giá trị trên các biến này cũng mất đi*) và thoát khỏi hàm. Nếu sau ***return*** có ***Biểu thức*** thì giá trị của biểu thức sẽ được tính, chuyển kiểu cho phù hợp với giá trị trả về và được gán cho hàm. Giá trị này sẽ ***có thể được sử dụng trong các hàm khác*** như giá trị đầu vào của chúng (*đây là một cách truyền thông tin giữa các hàm với nhau*) và nó chỉ có thể là một ***giá trị*** (*biến thường hoặc biến cấu trúc*) hoặc một ***địa chỉ (của biến thường hoặc biến cấu trúc)***.

- Như vậy để các hàm có thể trao đổi thông tin với nhau, ***ta chỉ có thể thực hiện theo một trong ba cách sau***:

+ *Sử dụng giá trị trả về của hàm (xem ví dụ 4-9, 4-10).*

+ Sử dụng cách truyền tham số theo địa chỉ (xem ví dụ 4-5).

Tuy nhiên việc *truyền tham số theo địa chỉ* nhiều khi sẽ dẫn đến các kết quả *logic* khác với suy luận thông thường. Sự khác nhau này còn gọi là *hiệu ứng lề*. Sau đây là một ví dụ điển hình về hiệu ứng lề khi sử dụng truyền tham số theo địa chỉ:

Ví dụ 4-6. Hiệu ứng lề khi sử dụng truyền tham số theo địa chỉ.

```
/* ***************************************************************************************** */
#include "stdio.h"
#include "conio.h"
int  Tang(int *); /* Nguyên mẫu của hàm */
/* ***************************************************************************************** */
int  main()
{
      int  x = 10;
      printf("Tong la %d", Tang(x) + Tang(x));
      getch() ;
      return 0 ;
}
/* ***************************************************************************************** */
int  Tang (int * a)
{
      ++a;
      return (a);
}
/* ***************************************************************************************** */
```

Khi thực hiện chương trình ta nhận được: Tong la 23

Rõ ràng bằng suy luận thông thường thì kết quả đúng phải là 22 (*vì khi x=10 thì Tang(x) cho giá trị 11*). Tuy nhiên, ở đây đã xảy ra hiệu ứng lề do truyền tham số cho hàm theo địa chỉ. Ta có thể giải thích như sau: ở lần gọi thứ nhất hàm *Tang(x)* đã thực sự làm giá trị của *x* tăng lên 1 (*tức là 11*), do đó ở lần gọi thứ hai giá trị của *x* thực sự sẽ là *12* và do đó tổng sẽ là *23*.

+ Sử dụng các biến toàn cục:

Vì các hàm đều có thể truy nhập và thay đổi giá trị trên các biến toàn cục, cho nên kết quả của hàm này có thể truyền sang hàm khác.

Đặc điểm: Không cần truyền tham số cho hàm thông qua các tham số hình thức (*vì các hàm xử lí trực tiếp trên các biến toàn cục*) cho nên đơn giản. Tuy nhiên khi viết chương trình ta nên hạn chế sử dụng biến toàn cục nếu không cần thiết (*vì dễ gây ra hiệu ứng lề không mong muốn, cấu trúc chương trình không sáng sủa và mất tính riêng tư của các hàm*).

Ví dụ 4-7. Trao đổi thông tin giữa các hàm thông qua biến toàn cục. Chương trình thực hiện tính tổng các bình phương của hai số.

```
/* ***************************************************************************************** */
#include "stdio.h"
#include "conio.h"
void  BinhPhuong(void); /* Nguyên mẫu của hàm */
void Tong(void);
int a, b, c, Dem=0; /* Các biến toàn cục dùng làm tham số cho các hàm*/
/* ***************************************************************************************** */
```

```
int main()
{
	Tong(); /* Kết quả của tổng lưu vào biến c */
	printf("\n Tong binh phuong cua hai so %d va %d la  %d", a, b, c);
	return 0 ;
}
/* ******************************************************************************** */
void BinhPhuong()
{
	printf("\nHay nhap gia tri cho so thu %d =  ", ++Dem);
	scanf("%d", &a);
	b=a*a; /* Hàm lưu giá trị bình phương tính được vào biến b */
	return;
}
/* ******************************************************************************** */
void Tong(void)
{
	int Tam ;
	BinhPhuong() ; /* Tính bình phương của số thứ nhất */
	Tam=b ; /* Lấy kết quả ra, nếu không giá trị gọi lần 2 sẽ ghi đè lên */
	BinhPhuong() ; /* Kết quả lần tính hai cũng lưu trong biến b */
	c=Tam+b ; /* Đưa kết quả của tổng vào biến c */
}
/* ******************************************************************************** */
```

4.3.3. Các cách truyền tham số cho hàm

Trong phần trước ta đã làm quen với cách xây dựng và hoạt động của hàm cũng như *cách truyền tham số cho hàm theo tham trị và theo địa chỉ*. Tuy nhiên các tham số thực được xét mới chỉ là các biến đơn giản có sẵn trong ngôn ngữ như ***int, float***... Trong phần này ta sẽ tiếp tục nghiên cứu các cách truyền tham số phức tạp hơn cho hàm như mảng, ma trận, cấu trúc...

1. Tham số thực là tên mảng một chiều

Khi tham số thực là tên mảng (*một chiều*) thì tham số hình thức tương ứng cần phải là một ***con trỏ*** có kiểu phù hợp với kiểu của các phần của tử mảng.

Ví dụ 4-8. Nếu tham số thực là một mảng kiểu số thực ***float MangThuc[50]*** thì tham số hình thức ***MangF*** tương ứng của hàm sẽ được khai báo theo một trong các cách sau đây:

float *MangF; Hoặc có thể khai báo như một mảng hình thức:

float MangF[];

Hai cách khai báo trên là tương đương nhau. Khi hàm bắt đầu làm việc thì giá trị hằng địa chỉ ***MangThuc*** của tham số thực sẽ được truyền cho tham số hình thức là con trỏ ***MangF***, nói cách khác con trỏ ***MangF*** sẽ chứa địa chỉ của phần tử đầu tiên trong mảng tham số thực. Do đó trong thân của hàm ta có thể dùng một trong những cách sau đây để truy nhập các phần tử của mảng ***MangThuc[i]***:

***(MangF+i) hoặc MangF[i]**

Ví dụ 4-9: Viết chương trình thực hiện việc nhập và cộng hai đa thức $\mathbf{P_n(x)}$ và $\mathbf{Q_m(x)}$. Kết quả được lưu trong mảng $\mathbf{C_{Max(m,n)}(x)}$. Đưa kết quả ra màn hình.

Để giải được bài toán này ta chọn cấu trúc dữ liệu kiểu mảng, mỗi phần tử thứ *i* (*0=<i<=Max(n,m)*) của mảng sẽ lưu trữ giá trị hệ số cho phần tử x^i của đa thức. Phần tử nào khuyết thì sẽ có hệ số bằng không. Chương trình được tổ chức thành các hàm nhập dữ liệu cho đa thức, cộng hai đa thức và đưa ra màn hình đa thức kết quả dưới dạng *Cx= C0*x^0 +C1*x^1 +...+Ck*x^k* với *k* là *Max(n,m)*.

```
/* ******************************************************************************** */
#include "stdio.h"
#include "conio.h"
#define Max(A,B) (A)>(B)?(A):(B)
#define MAX 30
int NhapMang(float *, char); /*Khai báo các nguyên mẫu của hàm*/
void HienThi(float *, int, char);
void Cong(float *, float*, float *, int);
/* ******************************************************************************** */
int main()
{
	int m, n;
	float Px[MAX], Qx[MAX], Cx[MAX]; /*Các tham số thực*/
	clrscr();
	m=NhapMang(Px, 'P');
	n=NhapMang(Qx, 'Q');
	Cong(Px, Qx, Cx, Max(n,m));
	HienThi(Cx, Max(m,n), 'C');
	getch();
	return 0;
}/* Kết thúc hàm main */
/* ******************************************************************************** */
int NhapMang(float *Pm, char ch)
{
	int n, m;
	printf("\nHay nhap bac cua da thuc %c\n", ch);
	scanf("%d", &n);
	for(m=0; m<=n; ++m)
	{
		printf("\nHay nhap he so %c[%d] cho phan tu %c[%d].X^%d: ", ch, m, ch, m, m);
		scanf("%f", (Pm+m));
	}
	for(m=n+1; m<MAX; ++m)
		Pm[m]=0;
	return n; /*Trả về kích thước mảng vừa nhập*/
}
/* ******************************************************************************** */
void Cong(float *px, float *qx, float*cx, int m)
{
	int i;
	for(i=0; i<=m; ++i)
		cx[i]= px[i]+qx[i];
	return;
}
/* ******************************************************************************** */
```

```
void HienThi(float *Pm, int m, char ch)
{
    int i, Dau; /* Dau =1 sẽ in ra dấu + */
    printf("\nDa thuc  ket qua se co dang nhu sau :\n");
    printf("%cx= ", ch);
    for(i=0; i<=m; ++i)
        if(Pm[i]!=0.0)
        {
                if(Pm[i]>0) Dau=1;
                else Dau=0;
                if(Dau)  printf("+");
                printf("%.1f*X^%d", Pm[i], i);
        }
     return;
}
/* ************************************************************************************************ */
```

Bài tập:

- Nhận xét gì về cách truyền tham số trong các hàm trên? Nếu không dùng tham số chỉ kích thước của mảng trong các hàm *Cong* và *HienThi* thì ta phải làm gì trong trường hợp này? Sửa lại chương trình.

- Tổ chức dữ liệu như trên có ưu và nhược điểm gì ? Có thể khắc phục được hay không ? Nếu được thì khắc phục như thế nào ?

- Câu lệnh

```
for(m=n+1; m<MAX; ++m)
    Pm[m]=0;
```

Trong hàm *NhapMang* có tác dụng gì?

2. Tham số thực là tên mảng hai chiều (ma trận)

Khi tham số thực là tên của một mảng hai chiều ***MaTran[30][40]*** chẳng hạn, thì vấn đề trở nên phức tạp hơn nhiều. Ta có hai cách khác nhau để thực hiện điều này.

Cách 1:

Dùng *tham số hình thức* là một con trỏ chứa được kiểu địa chỉ ***float[40]*** theo một trong các mẫu khai báo sau:

float (*Pf)[40]; hoặc
float Pf[][40]; /* Mẫu này chỉ dùng để khai báo tham số hình thức */

Trong thân hàm để truy nhập đến các phần tử của ma trận tham số ta có thể dùng ***Pf[i][j]*** (*vì Pf đã chứa địa chỉ của phần tử đầu tiên trong ma trận*).

Chú ý:

Theo cách này hàm chỉ có thể hoạt động được với các mảng hai chiều có *40* cột mà thôi (*số hàng tùy ý*).

Cách 2:

Dùng hai tham số là **float *Pf**; /* Chứa địa chỉ phần tử đầu tiên của ma trận */

và **int N**; /* Biểu thị số cột của của ma trận*/

Trong thân hàm, để truy nhập đến phần tử *MaTran[i][j]* ta dùng công thức *$*(Pf + i * N + j)$*.

Chú ý:

Theo cách 2, ma trận được quy về dạng véc tơ, do đó việc xử lí trong thân hàm sẽ phức tạp hơn cách 1. Tuy nhiên nó có thể dùng cho bất kì ma trận nào.

Ví dụ 4-10. Xét bài toán cộng hai đa thức $P_{mxn}(x,y)$ và $Q_{m1xn1}(x,y)$, kết quả được lưu trong đa thức $C_{Max(m,m1)xMax(n,n1)}(x,y)$. Ta có thể quy bài toán này về bài toán cộng hai ma trận bằng cách lựa chọn cấu trúc dữ liệu như sau: Mỗi đa thức $P_{mxn}(x,y)$ sẽ được lưu trữ trong một ma trận cấp ***mxn*** theo nguyên tắc hệ số của phần tử $x^i y^j$ được chứa trong phần tử ***dòng i và cột j của ma trận***. Về mặt tổ chức chương trình cũng tương tự như ví dụ 4-9. Một điều cần lưu ý là do hàm Nhap, Cong và HienThi làm việc với các ma trận chưa định trước do đó phải khai báo theo cách 2.

```
/* ***************************************************************************************** */
#include "stdio.h"
#include "conio.h"
#define Max(A,B) (A)>(B)?(A):(B)
#define MAXX  30
#define MAXY  40
typedef struct
{
     int m; /*Số dòng*/
     int n; /*Số cột*/
} BacMT;
BacMT  NhapMaTran(float *, int, char*);
void HienThi(float *, int, BacMT, char*); /* Tham số là một biến cấu trúc*/
void Cong(float *, float*, float *, int, BacMT);
/* ***************************************************************************************** */
int main()
{
     int N=MAXY;
     BacMT b1, b2, b3; /*Chứa bậc của các ma trận Pxy, Qxy và Cxy */
     float Pxy[MAXX][MAXY], Qxy[MAXX][MAXY], Cxy[MAXX][MAXY]; /* Các tham
     số thực */
     clrscr();
     b1= NhapMaTran((float*)Pxy, N, "Pxy"); /* Hàm trả về một cấu trúc chỉ bậc
     thực sự của ma trận vừa nhập */
     b2=NhapMaTran((float*)Qxy, N, "Qxy");
     /*Tìm cấp lớn nhất theo cả hai biến của Px và Qx*/
     b3.m=Max(b1.m, b2.m);
     b3.n=Max(b1.n, b2.n);
     Cong((float*)Pxy, (float*)Qxy, (float*)Cxy, N, b3);
     HienThi((float*)Cxy, N, b3, "Cxy");
     getch();
     return 0;
}/* Kết thúc hàm main */
/* ***************************************************************************************** */
BacMT  NhapMaTran(float *Pm, int N, char *Ch)
{
```

```
    BacMT b, i;
    for(i.m=0; i.m<MAXX; ++i.m)
        for(i.n=0; i.n<MAXY;++i.n)
            *(Pm+i.m*N+i.n)=0;
    printf("\nHay nhap bac cua da thuc  %s  theo bien x\n" , Ch);
    scanf("%d", &b.m);
    printf("\nHay nhap bac cua da thuc  %s  theo bien y\n", Ch);
    scanf("%d", &b.n);
    for(i.m=0; i.m<=b.m; ++i.m)
        for(i.n=0; i.n<=b.n; ++i.n)
        {
            printf("\nHay nhap he so %s[%d][%d] cho phan tu  %s[%d][%d]*X^%d*Y^%d: ",
            Ch, i.m, i.n, Ch, i.m, i.n, i.m, i.n);
            scanf("%f", Pm+i.m*N+ i.n);
        }
    return b; /*Trả về kích thước thực của ma trận vừa nhập*/
}
/* ******************************************************************************************* */
void Cong(float *pxy, float *qxy, float*cxy, int N, BacMT b)
{
    BacMT i;
    for(i.m=0; i.m<=b.m; ++i.m)
        for(i.n=0; i.n<=b.n; ++i.n)
        {
            *(cxy+i.m*N+i.n)= *(pxy+i.m*N+i.n)+ *(qxy+i.m*N+i.n);
        }
    return;
}
/* ******************************************************************************************* */
void HienThi(float *Pm, int N, BacMT b, char *Ch)
{
    BacMT i;
    int  Dau;
    printf("\nDa thuc  ket qua se co dang nhu sau :\n");
    printf("%s= " , Ch);
    for(i.m=0; i.m<=b.m; ++i.m)
        for(i.n=0; i.n<=b.n; ++i.n)
        {
            if(*(Pm+i.m*N+ i.n)!=0.0)
            {
                if(*(Pm+i.m*N+ i.n)>0.0)  Dau=1;
                else Dau=0;
                if(Dau)  printf("+");
                printf("%.1f*X^%d*Y^%d", *(Pm+i.m*N+i.n), i.m, i.n);
            }
        }
    return ;
}
/* ******************************************************************************************* */
```

Chú ý: Nếu tham số hình thức hay giá trị trả về là một biến hay con trỏ cấu trúc thì cấu trúc đó phải được khai báo bằng từ khoá *typedef* trước khi sử dụng.

4.4. CON TRỎ TỚI HÀM (CON TRỎ HÀM)

4.4.1. Khái niệm và cách sử dụng

Trong khi viết chương trình đôi khi ta cần thiết kế các hàm mà tham số thực trong lời gọi đến nó lại là tên của một hàm khác. Ví dụ để viết một hàm tính tích phân xác định cho hàm số *f(x)* theo một công thức gần đúng nào đó, ta cần truyền cho nó một đối số chính là hàm *f(x)* đó... Để có thể thực hiện được điều này ta cần dùng đến con trỏ hàm. Con trỏ hàm là một loại con trỏ đặc biệt dùng để chứa địa chỉ của một hàm và được khai báo theo cú pháp sau đây:

KiểuTrảVề (*TênHàm)(Danh sách kiểu tham số); /*Khai báo con trỏ hàm có kiểu là **KiểuTrảVề** */

KiểuTrảVề (*TênHàm[Kích thước])(Danh sách kiểu tham số); /*Khai báo cho mảng con trỏ hàm*/

Ví dụ 4-11. Để khai báo ra một con trỏ hàm có tên là *f*, có kiểu là ***float*** và tham số hình thức có kiểu là ***int*** ta viết như sau: ***float (*f)(int);***

Để khai báo ra một con trỏ hàm có tên là *g*, có kiểu là ***float*** và các tham số hình thức có kiểu lần lượt là ***int và float*** ta viết như sau: ***float (*g)(int, float);***

Để khai báo ra một mảng con trỏ hàm gồm *30* phần tử có tên là ***Mf***, có kiểu là ***float*** và các tham số hình thức có kiểu lần lượt là ***int và double*** ta viết như sau:

float (*Mf[30])(int, double);

Một con trỏ hàm trước khi sử dụng phải được gán cho một tên hàm xác định. Để phép gán có ý nghĩa thì *kiểu hàm* và *kiểu con trỏ hàm* phải tương thích. Phép gán có thể được thực hiện ngay trong lúc khai báo hoặc sau khi khai báo. Sau phép gán, ta có thể dùng tên con trỏ hàm thay cho tên hàm. Ví dụ sau đây sẽ cho ta thấy rõ điều đó:

```
...
double TinhMax(double x, double y) /* Định nghĩa hàm tính số lớn nhất */
{
        return (x>y?x:y);
}
...
/*Khai báo và gán tên hàm cho con trỏ hàm*/
double (*PMax)(double, double)=TinhMax;
int main()
{
        printf("\nMax= %f", PMax(4.2,6.5));
}
```

4.4.2. Đối con trỏ hàm

Khi tham số thực là ***tên của một hàm*** nào đó trong lời gọi hàm thì tham số ***hình thức*** cho hàm đó phải là *một con trỏ hàm.*

Khi đó, trong thân hàm ta có thể sử dụng các tham số con trỏ hàm theo một trong ba cách sau:

TênConTrỏHàm(Danh sách tham số thực của hàm được trỏ bởi nó);

(TênConTrỏHàm)(Danh sách tham số thực của hàm được trỏ bởi nó);

(*TenConTroHam)(Danh sách tham số thực của hàm được trỏ bởi nó);

Ví dụ 4-12. Viết hàm tính tích phân của hàm một *f(x)* trên đoạn [a,b] theo phương pháp hình thang bằng cách chia đoạn [a,b] thành *1000* khoảng có độ dài như nhau. Sau đó dùng hàm vừa xây dựng tính và đưa kết quả ra màn hình tích phân của các hàm sau đây:

- $f(x)=\sin x$ trong khoảng $[0,\pi/2]$
- $f(x)=\cos x$ trong khoảng $[0,\pi/2]$
- $f(x)=e^x$ trong khoảng [0,1.0]
- $f(x)=(e^x -2\sin x^2)/(1+x^n)$ trong khoảng [-1,2,3.5]

Giải: Rõ ràng ở đây ta phải sử dụng hàm tính tích phân có đối là một con trỏ hàm để có thể gán các hàm khác nhau cho nó. Chương trình được viết như sau:

```
/* ************************************************************************************** */
#include "stdio.h"
#include "conio.h"
#include "math.h"
#define PI  3.14
double TichPhan(double (*f)(double), double a, double b);
double g(double);
/* ************************************************************************************** */
int main()
{
      printf("\nF1= %f", TichPhan(sin, 0, PI/2));
      printf("\nF2= %f", TichPhan(cos, 0, PI/2));
      printf("\nF3= %f", TichPhan(exp, 0, 1.0));
      printf("\nF4= %f", TichPhan(g, -1.2, 3.5));
      getch();
      return 0;
}
/* ************************************************************************************** */
double TichPhan(double (*f)(double), double a, double b)
{
      int i, n=1000;
      double s, h= (b-a)/n;
      s= (f(a)+f(b))/2;  /* Dùng đối con trỏ hàm trong thân hàm */
      for(i=1; i<n; ++i)
      {
           s+= f(a+i*h);
           return (h*s);
      }
}
/* ************************************************************************************** */
double g(double x)
{
      double s;
      s=(exp(x)- 2*sin(x*x)) / (1+pow(x,4));
      return s;

}
```

Ví dụ 4-13. Xây dựng hàm sắp xếp tổng quát cho một dãy *n* đối tượng đặt trong vùng nhớ do con trỏ *Buffer* kiểu *void* trỏ tới. Độ dài của đối tượng là *size* bytes. Tiêu chuẩn sắp xếp cho theo hàm *SoSanh*. Dùng hàm đó để sắp xếp một dãy số thực theo tiêu chuẩn tăng dần và một dãy số nguyên theo tiêu chuẩn giảm dần.

```
/* ******************************************************************************************** */
#include "stdio.h"
#include "conio.h"
#include "math.h"
#include "mem.h"
#include "alloc.h"
#define MAX 20
void SapXep(void *Buf, int size, int n, int (*ss)(void*,void*));
int Tang(void*,void*);
int Giam(void*, void*);
int NhapThuc(float *);
int NhapNguyen(int*);
/* ******************************************************************************************** */
int main()
{       int j, SoPhanTu, Y[MAX];
        float F[MAX];
        SoPhanTu=NhapThuc(F);
        SapXep(F, sizeof(float), SoPhanTu,Tang);
        clrscr();
        printf("\nDay so thuc da duoc sap xep la:\n");
        for(j=0; j<=SoPhanTu; ++j)
        {
                printf("%10.2f", F[j]);
        }
        getch();
        SoPhanTu=NhapNguyen(Y);
        SapXep(Y, sizeof(int), SoPhanTu, Giam);
        clrscr();
        printf("\nDay so nguyen da duoc sap xep la:\n");
        for(j=0; j<=SoPhanTu; ++j)
        {
                printf("%10d", Y[j]);
        }
        getch();
        return 0;
}
/* ******************************************************************************************** */
int  NhapThuc(float *Pf)
{
        int Dem=0;
        printf("\nHay nhap cho mang so thuc :");
        while(1)
        {
                printf("\nHay nhap cho F[%d]= ", Dem);
```

```
            scanf("%f", Pf+Dem);
            if(Pf[Dem]==0.0)  break;
            ++Dem;
      }
      return (Dem-1);
}
/* ******************************************************************************************* */
int  NhapNguyen(int *Pi)
{
      int Dem=0;
      printf("\nHay nhap cho mang so nguyen :");
      while(1)
      {
            printf("\nHay nhap cho Y[%d]= ", Dem);
            scanf("%d", Pi+Dem);
            if(Pi[Dem]==0)  break;
            ++Dem;
      }
      return (Dem-1);
}
/* ******************************************************************************************* */
int Tang(void*u, void*v)
{
      return (*((float*)u)<=*((float*)v));
}
/* ******************************************************************************************* */
int Giam(void*u, void*v)
{
      return (*((int*)u)>=*((int*)v));
}
/* ******************************************************************************************* */
void SapXep(void *Buf, int size, int n, int (*ss)(void*,void*))
{
      void *tg; char *p; int i,j;
      p=(char*)Buf;/*truy nhập vùng nhớ theo từng byte*/
      tg=(char*)malloc(size);
      /* Sắp xếp các đối tượng*/
      for(i=0; i<n; ++i)
            for(j=n; j>=i+1; --j)
                  if(!ss(p+i*size, p+j*size))
                  {
                        /*Đổi chỗ hai đối tượng cho nhau*/
                        memcpy(tg, p+i*size, size); /*Copy size byte từ địa chỉ p+i*size
                        vào biến trỏ tg, hàm này được khai báo trong mem.h*/
                        memcpy(p+i*size, p+j*size, size);
                        memcpy(p+j*size, tg, size);
                  }
}
/* ******************************************************************************************* */
```

4.5. ĐỆ QUY VÀ VIẾT CHƯƠNG TRÌNH KIỂU ĐỆ QUY

4.5.1. Khái niệm về đệ quy

1. Đối tượng đệ quy và định nghĩa đệ quy

Một đối tượng được gọi là đệ quy nếu nó bao gồm chính nó như một bộ phận hoặc được định nghĩa dưới dạng của chính nó.

Ví dụ 4-14. Sau đây là một số các định nghĩa đệ quy trong toán học

**** Định nghĩa của số tự nhiên***

a) 1 là một số tự nhiên.

b) x là một số tự nhiên nếu như x-1 cũng là một số tự nhiên.

**** Định nghĩa n!***

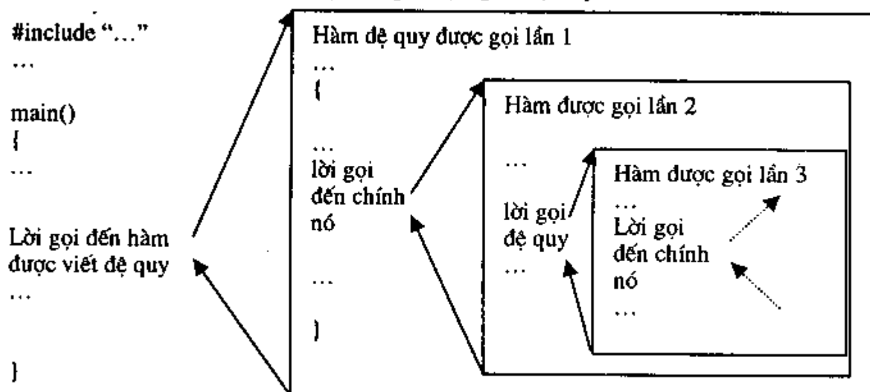
a) 0!=1

b) n!=n(n-1)!*

2. Giải thuật đệ quy và hàm đệ quy

Nếu lời giải của một bài toán T được thực hiện bằng lời giải của một bài toán T' có dạng giống T thì đó là một lời giải đệ quy. Giải thuật tương ứng với một lời giải đệ quy thì gọi là *giải thuật đệ quy*. Một hàm được viết theo một giải thuật đệ quy thì được gọi là *hàm đệ quy*. Thông thường một hàm đệ quy phải có một lời gọi đến chính nó trong thân hàm hoặc được gọi một cách gián tiếp qua một hàm khác. Khi một hàm gọi đệ quy đến chính nó thì mỗi lần gọi máy sẽ tạo ra một tập các biến cục bộ mới *độc lập với các lời gọi trước đó* và các biến này sẽ mất đi khi bản hàm được gọi đó kết thúc. Có bao nhiêu lời gọi hàm được thực hiện thì cũng có bấy nhiêu lần hàm được kết thúc và ***trả giá trị về cho nơi đã gọi nó***. Chính vì thế trong khi viết chương trình dạng đệ quy, cần phải hết sức chú ý đến việc *truyền tham số và giá trị trả về của hàm*. Nếu không có thể dẫn đến treo máy hoặc cho kết quả sai. Hình vẽ sau sẽ mô tả khá đầy đủ cho các hoạt động của hàm viết theo kiểu đệ quy.

Hàm được gọi lần ***i*** sẽ trả giá trị về cho hàm đã gọi nó ở lần ***i-1***. Các bản hàm ở các lần gọi khác sẽ không thể truy nhập được giá trị này.



Hình 4.1. Hình ảnh minh hoạ hoạt động của các lời gọi đệ quy.

Bài tập:

Có nhận xét gì về các mặt sau, khi chương trình viết theo dạng đệ quy?

- *Về tốc độ thực hiện*
- *Cấu trúc chương trình*
- *Cách thức thực hiện*

4.5.2. Thiết kế giải thuật đệ quy, viết hàm kiểu đệ quy

Việc thiết kế giải thuật đệ quy cũng tuân thủ theo các nguyên tắc đã nêu trong mục 4.1. Ở đây, ta cần xét các trường hợp sau:

a) Trường hợp 1: Bản thân các bài toán đang xét đã được định nghĩa ở dạng đệ quy. Khi đó việc xây dựng các hàm đệ quy không mấy khó khăn (*chỉ là vấn đề mã hóa định nghĩa đó bằng ngôn ngữ lập trình*).

Ví dụ 4-15. Xét bài toán tính *n!* ở trên. Từ định nghĩa của nó ta có thể xây dựng hàm đệ quy như sau:

```
long GiaiThuat( int n )
{
	if(n==0)
		return 1;
	else
		return (n*GiaiThuat( n-1)); /* Chứa lời gọi đến chính nó */
}
```

b) Trường hợp 2. Với những bài toán chưa có định nghĩa đệ quy ta phải đi tìm định nghĩa đệ quy cho nó (*nếu có thể*) rồi tiến hành như trường hợp 1.

Ví dụ 4-16. Xét bài toán cổ tính số lượng các cặp thỏ ở tháng thứ ***n*** biết rằng, ban đầu ta chỉ có một cặp thỏ con. Các cặp thỏ sẽ sinh sản theo quy luật sau đây (*dãy này còn được gọi là dãy số FIBONACCI*):

- *Chỉ hai tháng sau khi ra đời một cặp thỏ mới để lần đầu ra một cặp thỏ con (một đực, một cái).*
- *Sau khi đã sinh con thì mỗi tháng sau đó chúng sẽ sinh ra một cặp thỏ con mới.*

Giả thiết rằng các con thỏ không bao giờ chết.

Từ bài toán trên ta có nhận xét sau:

Nếu ở tháng thứ n-1 mà tất cả các cặp thỏ đều có thể sinh đẻ thì ở tháng thứ n ta sẽ có số cặp thỏ gấp đôi ở tháng thứ n-1. Tuy nhiên, ở tháng thứ n lại chỉ có những cặp thỏ ở tháng thứ n-2 sinh đẻ được. Có nghĩa là số cặp thỏ ở tháng thứ n sẽ tăng lên đúng bằng số cặp thỏ đã có ở tháng thứ n-2. Nếu giả sử rằng:

Hàm Fibonacci(n) cho ta số cặp thỏ ở tháng thứ n thì
hàm Fibonacci(n-1) sẽ cho ta số cặp thỏ ở tháng thứ n-1 và
hàm Fibonacci(n-2) cho ta số cặp thỏ ở tháng thứ n-2

Như phân tích ở trên thì ta có công thức đệ quy sau đây:

Fibonacci(n) = Fibonacci(n-1)+ Fibonacci(n-2)

Với *Fibonacci(1)=1*

*Fibonacci(2)=1 /*tháng đầu và tháng tiếp theo vẫn chưa sinh*/*

Hai trường hợp này người ta còn gọi là trường hợp suy biến hay điều kiện đầu và việc tính cho các trường hợp khác phải dựa trên các trường hợp này.

Từ công thức đệ quy vừa tìm được ta có thể viết hàm đệ quy như sau:

```
long Fibonacci(int n)
{
	if((n==1)||(n==2))
		return 1;
	else
		return (Fibonacci(n-1)+Fibonacci(n-2));
}
```

Từ việc thiết kế giải thuật và viết cho các hàm đệ quy ta rút ra các đặc điểm sau đây cho một hàm đệ quy:

(1) Trong hàm phải có lời gọi đến chính nó (trực tiếp) hoặc gọi đến một hàm khác mà hàm này lại chứa một lời gọi đến nó (gián tiếp).

(2) Sau mỗi lần gọi hàm kích thước của bài toán theo một nghĩa nào đó phải thu nhỏ hơn trước, tiến dần tới trường hợp đặc biệt gọi là trường hợp suy biến.

(3) Trong định nghĩa đệ quy bao giờ cũng phải tồn tại một trường hợp đặc biệt (tính khác các trường hợp khác) gọi là trường hợp suy biến mà ở đó giải thuật sẽ kết thúc và trả kết quả về cho lời gọi trước đó. Đây là đặc điểm vô cùng quan trọng nó quy định tính dừng của thuật toán, nếu không ta có thể rơi vào vòng lặp vô tận.

Do vậy khi viết các hàm đệ quy bao giờ cũng phải kiểm tra trường hợp suy biến trước khi đưa ra lời gọi đệ quy.

4.7. CÂU HỎI VÀ BÀI TẬP

1. Hãy phân biệt việc truyền tham số theo tham trị và truyền theo địa chỉ trong quá trình trao đổi dữ liệu với các hàm? cho ví dụ.

2. Phân biệt tham số hình thức và tham số thực của hàm ? Cho ví dụ.

3. Sự khác nhau giữa tổ chức chương trình trong ngôn ngữ lập trình C và Pascal là gì? Ưu và nhược của từng loại cấu trúc đó ?

4. Phân biệt giá trị trả về và danh sách tham số?

5. Hãy trình bày các cách trao đổi thông tin khác nhau giữa các hàm trong một chương trình viết bằng ngôn ngữ lập trình C ?

6. Viết lại ví dụ 4-9 bằng cách tổ chức dữ liệu theo kiểu danh sách nối đơn.

7. Viết lại ví dụ 4-10 bằng cách tổ chức dữ liệu theo kiểu danh sách.

8. Hãy viết chương trình tính giá trị của các hàm sau rồi liệt kê kết quả (*theo từng cột cho mỗi hàm*) ra màn hình:

f1=x^2; f2=sinx; f3=cosx; f4=e^x ; và f5= sqrt(x) với x lấy các giá trị từ 1.0 đến 10.0 theo bước nhảy là 0.5

9. Viết chương trình thực hiện nhân ma trận vuông cấp *n* với véc tơ X (*n phần tử*).

10. Viết chương trình thực hiện hoán vị một ma trận chữ nhật *mxn*

11*. Viết chương trình tính các giá trị nhỏ nhất và lớn nhất của hàm số *f(x)* trên đoạn *[a,b]*, với f cho bất kì.

12. Tìm một nghiệm của phương trình f(x)=0 trên đoạn *[a,b]* với giả thiết f(x) liên tục trên đoạn đã cho và *f(a)f(b)<0*

13*. Có *n* người làm *n* việc. Biết rằng nếu người *i* làm việc *j* thì đạt hiệu quả *a[i][j]*. Hãy lập phương án phân công mỗi người một công việc sao cho tổng hiệu quả là lớn nhất.

14. Viết chương trình nhập tọa độ *n* điểm trong không gian ($x_i y_i z_i$) rồi tìm tọa độ các đỉnh của một hình hộp có các cạnh song song với các trục tọa độ và chứa tất cả các điểm trên.

15. Viết hàm thực hiện việc đổi tọa độ của hai véc tơ cho nhau.

16. Viết hàm thực hiện việc hoán vị giá trị các phần tử tương ứng của hai ma trận có cùng kích thước A và B. Đưa kết quả ra màn hình.

17. Viết hàm để viết một câu vào vị trí (x,y) và hàm xóa câu từ vị trí (x,y) trở đi, trong đó (*1<=x<=80;1<=y<=25*). Dùng các hàm đó viết một chương trình cho dòng chữ :

"Đại Học Bách Khoa Hà Nội" chạy trên màn hình từ trái qua phải hoặc chạy trên đường chéo màn hình tùy chọn.

18. Viết chương trình nhập một văn bản *T*, đếm các kí tự khác nhau và số lần xuất hiện các kí tự đó trong *T*. Tính tỉ số giữa số lần xuất hiện từng kí tự với độ dài văn bản *T*. Đưa kết quả ra màn hình.

19*. Viết chương trình thực hiện việc đổi một số từ cơ số *10* sang cơ số *16*. Đưa kết quả ra màn hình dưới dạng số hệ *16*.

20. Viết các hàm thực hiện các thao tác bổ sung, loại bỏ trên danh sách móc nối ?

21. Viết các hàm thực hiện các thao tác bổ sung, loại bỏ trên danh sách tuyến tính ?

22. Viết các hàm thực hiện các thao tác bổ sung, loại bỏ trên hàng đợi?

23. Viết các hàm thực hiện các thao tác bổ sung, loại bỏ trên ngăn xếp?

Chương 5

TỆP VÀ CÁC THAO TÁC VÀO RA

MỤC TIÊU CỦA CHƯƠNG NÀY

- *Hiểu đặc điểm và bản chất của việc lưu trữ dữ liệu trên các tệp tin*
- *Hiểu cấu trúc của tệp tin cũng như nguyên lý hoạt động khi tiến hành thao tác trên tệp tin*
- *Hiểu và biết cách viết chương trình xử lí tệp truy nhập tuần tự*
- *Hiểu và biết cách viết chương trình xử lí tệp truy nhập ngẫu nhiên*

5.1. CÁC KHÁI NIỆM VÀ CÁC ĐẶC TRƯNG KHI LƯU TRỮ THÔNG TIN TRÊN TỆP

Việc lưu dữ liệu trong bộ nhớ cho các biến, mảng, cấu trúc... mà ta đã làm quen từ đầu giáo trình cho đến thời điểm này chỉ là tạm thời, tất cả các dữ liệu như vậy sẽ bị mất khi chương trình kết thúc. Để có thể lưu giữ được dữ liệu lâu dài và có thể mang đi từ nơi này sang nơi khác, người ta sử dụng một cách lưu trữ khác đó là *lưu trữ ở bộ nhớ ngoài* (*đĩa từ, băng từ, đĩa CD*...). Việc quản lí dữ liệu theo cách lưu trữ này được dựa trên việc quản lí các tệp (*hay còn gọi là các file*). Trong chương này chúng ta sẽ tìm hiểu cấu trúc, đặc điểm cũng như các thao tác xử lí trên tệp, cả với tệp truy nhập tuần tự cũng như tệp truy nhập ngẫu nhiên.

Hệ thống phân cấp các tệp trong MS-DOS

Hệ thống các tệp trong MS-DOS mô tả cách các tệp được tổ chức và quản lí bởi hệ điều hành. MS-DOS dùng hệ thống phân cấp các tệp theo cấu trúc hình cây trong đó có gốc là thư mục gốc, tiếp dưới là các tệp hoặc các thư mục con, mỗi một thư mục có thể có các tệp và các thư mục khác nằm dưới nó.

Đường dẫn để truy nhập các tệp

Các tệp trong MS-DOS được xác định bởi đường dẫn của nó. Đó là một xâu với 4 thành phần: ***Tên ổ đĩa, tên các thư mục, tên tệp*** và ***phần mở rộng***. Sau đây là một ví dụ về đường dẫn đến tệp *text.txt*: "C:\LapTrinh\TC2\BaiTap\text.txt". Căn cứ vào đường dẫn này ta có thể truy xuất đến một tệp cần xét.

Các cách thức thao tác trên tệp tin và các đặc trưng

Để có thể thực hiện được các thao tác trên tệp tin, trong ngôn ngữ lập trình C người ta tiến hành thông qua các hàm thư viện. Các hàm này được chia làm hai nhóm đó là ***nhóm các hàm cấp 1*** (*còn gọi là các hàm truy nhập ở mức thấp, chúng được khai báo trong các tệp tin thư viện io.h, fcntl.h, sys/stat.h và dos.h*) và ***nhóm các hàm cấp 2*** (*được khai báo trong tệp tin thư viện stdio.h*). Các hàm cấp 1 thực hiện việc đọc ghi như DOS. Chúng có đặc trưng là:

+ *Không có dịch vụ xuất/nhập riêng cho từng kiểu dữ liệu mà chỉ có các thao tác trên các byte.*

+ *Mỗi tệp khi hoạt động đều có một thẻ gọi là thẻ file. Mọi thao tác xuất nhập đến tệp của các hàm đều thông qua thẻ này.*

Các hàm cấp 2 được xây dựng dựa trên các hàm cấp 1 nên dễ sử dụng và nhiều chức năng hơn. Chúng có đặc điểm là:

+ Có các dịch vụ xuất/nhập cho từng kiểu dữ liệu (có các hàm thao tác trên các số nguyên, kí tự, cấu trúc...).

+ Các thao tác xuất/nhập được thông qua một vùng đệm trung gian (không được đọc/ghi trực tiếp) nhằm làm tăng tốc độ xuất/nhập giảm bớt số lần truy nhập đĩa từ. Khi đọc, dữ liệu được lấy từ vùng đệm và chỉ khi vùng đệm đã trống rỗng máy mới truy nhập đĩa để lấy dữ liệu đưa vào vùng đệm cho các lần đọc tiếp sau. Khi ghi dữ liệu vào tệp, thông tin cũng được cất vào vùng đệm và chỉ khi vùng đệm đầy dữ liệu mới được đẩy lên đĩa. Chính vì vậy khi làm việc trên tệp ta cần chú ý đến thao tác vét vùng đệm để tránh mất mát thông tin.

+ Các hàm cấp 2 làm việc với tệp thông qua một con trỏ gọi là con trỏ tệp đó là một con trỏ có kiểu là cấu trúc FILE (cấu trúc này đã được định nghĩa trong stdio.h). Mọi thao tác trên tệp sẽ được tiến hành thông qua con trỏ này.

Chính vì tính *đơn giản, đủ và dễ sử dụng* mà các hàm cấp 2 thường được ưa dùng hơn so với các hàm cấp 1. Trong giáo trình này chủ yếu đề cập đến các thao tác trên tệp bằng các hàm cấp 2.

Về mặt cấu trúc, mỗi một tệp tin (*dù được xây dựng bằng cách nào*) đều được tổ chức dưới dạng một ***danh sách tuyến tính*** (*do đó nó có các tính chất của của danh sách tuyến tính*) và bao gồm một dãy các bytes có giá trị từ 0 đến 255, số byte của dãy này chính là độ dài của tệp. Một giá trị đặc biệt (*khác các giá trị khác*) đánh dấu sự kết thúc của tệp đó là giá trị -1, giá trị này được định nghĩa cho dấu hiệu kết thúc tệp *EOF*. Trong quá trình đọc tệp tin, khi gặp *EOF* máy sẽ coi là đã kết thúc tệp và thao tác đọc dừng lại. Khi đó hàm ***feof()*** sẽ trả về một giá trị khác không (*giá trị 1*).

5.2. CÁC CHẾ ĐỘ THAO TÁC TRÊN TỆP TIN

Trong ngôn ngữ lập trình C, khi sử dụng các hàm cấp 2 người ta phân biệt hai chế độ thao tác trên các tệp tin đó là chế độ xuất/nhập theo kiểu nhị phân và chế độ xuất/nhập theo kiểu văn bản. Mỗi chế độ có những đặc điểm riêng mà trong khi ứng dụng, tùy theo từng hoàn cảnh cụ thể ta sẽ lựa chọn chế độ nào cho phù hợp.

Xuất/nhập kiểu nhị phân là kiểu xuất/nhập mà trong đó dữ liệu được ghi/đọc trên tệp theo các bytes nhị phân như chúng nằm trong bộ nhớ trong (*dữ liệu không bị biến đổi gì*).

Còn xuất/nhập kiểu văn bản là kiểu xuất/nhập trong đó có sự chuyển đổi đối với một số kí tự đặc biệt để cho phù hợp với các tệp văn bản của DOS. Các kí tự đặc biệt đó là kí tự chuyển dòng có mã là 10 (*mã ASCII*) và kí tự mã 26. Một điểm khác biệt nữa của chế độ làm việc này với chế độ làm việc theo kiểu nhị phân là cách thức lưu trữ *dữ liệu kiểu số*. Các số trong chế độ này sẽ được lưu trữ dưới dạng một chuỗi các kí tự. Đối với các kí tự còn lại thì hai chế độ *xuất/nhập* làm việc giống nhau.

Trong chế độ xuất/nhập kiểu văn bản khi ghi, một kí tự mã *10* sẽ được chuyển thành hai kí tự là kí tự mã *13* và kí tự mã *10*. Khi đọc, hai kí tự mã *13* và *10* liên tiếp trên tệp sẽ được hợp lại còn một kí tự mã *10* như nguyên thủy. Điều này cốt để phù hợp với các văn bản của DOS vì các văn bản này mỗi dòng sẽ được kết thúc bằng hai mã *13* và *10*. Đối với mã *26* (*kí tự 1A trong hệ 16*) là mã đánh dấu sự

kết thúc của các tệp trong một số hệ soạn thảo như C, Pascal... thì khi đọc, nếu gặp kí tự có mã là *26* hoặc *EOF* thì ta đều nhận được một mã kết thúc tệp *EOF* với giá trị bằng -1 và hàm *feof* sẽ cho giá trị khác không (*đúng*). Do vậy, nếu bằng cách nào đó ta đưa được mã *26* vào một vị trí ở giữa của tệp tin thì khi mở tệp tin theo kiểu văn bản, đọc đến mã này chương trình đọc sẽ dừng hẳn vì lúc đó hàm đọc sẽ phát sinh giá trị -1 (*mã EOF*) để báo rằng đã kết thúc tệp tin.

Khi thao tác với các số, nên làm việc theo chế độ nhị phân vì các lí do sau:

*- Nếu ta đã ghi dữ liệu số ở dạng nhị phân sau đó lại đọc trong chế độ văn bản thì có thể sẽ có những mã **26** hoặc mã **10** tồn tại sẵn trong tệp làm cho dữ liệu đọc ra có thể sai **lệnh rất** nhiều. Do đó cần lưu ý là nếu một tệp đã được ghi theo chế độ nào thì nên **đọc lại** theo chế độ đó.*

*- Khi một **dữ liệu** số được ghi trong chế độ văn bản sẽ gây ra tốn bộ nhớ do nó sẽ lưu số đó **dưới** dạng các kí tự liên tiếp nhau. Ví dụ khi ghi số nguyên 2003 theo chế độ **văn bản thì** thay vì nó được lưu trữ trong 2 byte (với số nguyên) thì nó lại được ghi trong **4** byte (cho bốn kí tự '2', '0', '0' và '3').*

Trong mỗi chế độ trên, ta lại có thể làm việc theo kiểu tuần tự hoặc ngẫu nhiên. Như đã được đề cập trong mục *5.1*, tệp chính là một dạng danh sách tuyến tính có kích thước lớn được lưu trữ ở bộ nhớ ngoài. Do đó khi thực hiện các thao tác trên tệp (*đọc, ghi, bổ sung*...) cần phải có một cơ chế để đánh dấu sự bắt đầu và kết thúc tệp (*vai trò giống như con trỏ Dau và con trỏ Cuoi trong danh sách tuyến tính*). Để đánh dấu sự kết thúc tệp người ta dùng kí tự có mã 26 (*đối với một số trình soạn thảo văn bản*) hoặc dấu hiệu *EOF* (*đã được định nghĩa trong stdio.h*) có mã là -1 cho các loại tệp khác. Bắt đầu của một tệp bao giờ cũng từ byte thứ 0 và trong khi hoạt động luôn tồn tại một con trỏ gọi là *con trỏ chỉ vị* dùng để xác định vị trí hiện thời của tệp đang được phép truy xuất và các thao tác ghi/đọc chỉ diễn ra tại vị trí hiện thời của *con trỏ chỉ vị* trong tệp đó. Con trỏ chỉ vị có thể di chuyển từ byte 0 (*đầu tệp*) cho đến cuối tệp (*gặp dấu hiệu EOF*). Khi một tệp được mở để làm việc trong các kiểu đọc/ghi bình thường thì con trỏ chỉ vị luôn ở vị trí đầu của tệp tin (*byte 0*), còn nếu nó được mở theo kiểu để có thể *ghi bổ sung* (*kiểu a*) thì con trỏ chỉ vị sẽ nằm ở *cuối tệp* để cho phép ghi tiếp dữ liệu vào tệp (*các kiểu mở tệp sẽ được đề cập trong mục 5.3.1*). Sau khi một thao tác ghi/đọc đã được thực hiện xong, con trỏ chỉ vị sẽ tự động di chuyển đến phần tử tiếp theo trong tệp (*di chuyển về hướng cuối tệp một số byte đúng bằng số byte đã đọc/ghi*). Như vậy, việc xuất/nhập dữ liệu được tiến hành tuần tự theo chiều từ đầu đến cuối tệp tin. Cách thức làm việc này được gọi là *truy xuất tệp tin theo kiểu tuần tự*. Tuy nhiên, trong ngôn ngữ lập trình C cũng cho phép dùng các hàm để *di chuyển con trỏ chỉ vị* đến một vị trí bất kì trong tệp, phục vụ cho các thao tác cập nhật tệp tin (*bổ sung, sửa chữa*...). Cách thức làm việc này được gọi là *truy xuất tệp tin theo kiểu ngẫu nhiên* và sẽ được đề cập trong một mục riêng.

5.3. CÁC THAO TÁC CƠ BẢN TRÊN TỆP TIN

5.3.1. Mở đóng tệp, xóa vùng đệm và kiểm tra lỗi

Đây là các thao tác thường xuyên và quyết định trong quá trình làm việc với các tệp tin. Muốn thực hiện được các thao tác (*đọc, ghi, cập nhật*...) trên một tệp tin nào đó, trước đó phải dùng hàm mở tệp tin đó theo cú pháp như sau:

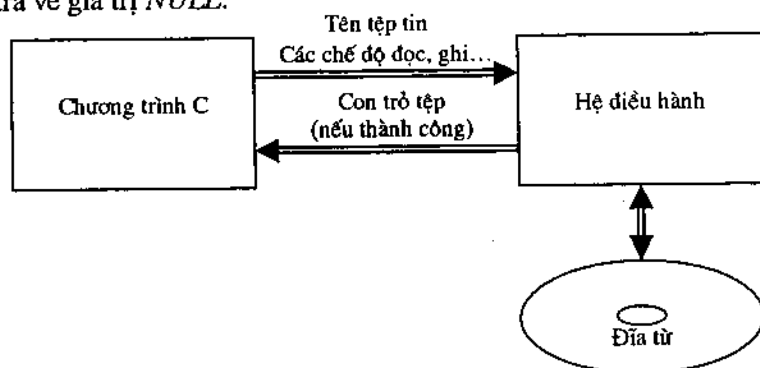
```
FILE *fopen(const char* TênTệp, const char * Kiểu);
```

Trong đó tham số thứ nhất là tên của tệp cần mở, còn tham số thứ hai là kiểu truy xuất. Kiểu truy xuất được cho trong bảng dưới đây:

Bảng 5.1. Bảng các kiểu truy xuất tệp tin cho hàm fopen

Kiểu	Chức năng
"r" ; "rt"	Mở một tệp chỉ đọc theo kiểu văn bản, sẽ có lỗi nếu tệp không tồn tại.
"w" ; "wt"	Mở một tệp mới để ghi trong chế độ văn bản, nội dung của tệp bị xoá nếu nó đã tồn tại.
"a" ; "at"	Mở một tệp để ghi bổ sung trong chế độ văn bản, nếu tệp chưa tồn tại nó sẽ được tạo mới.
"rb"	Mở một tệp để đọc trong chế độ nhị phân, sẽ có lỗi nếu tệp không tồn tại.
"wb"	Mở một tệp mới để ghi trong chế độ nhị phân, nội dung của tệp bị xoá nếu nó đã tồn tại.
"ab"	Mở một tệp để ghi bổ sung trong chế độ nhị phân, nếu tệp chưa tồn tại nó sẽ được tạo mới.
"r+" ; "r+t"	Mở một tệp để đọc và ghi trong chế độ văn bản, sẽ có lỗi nếu tệp không tồn tại.
"w+" ; "w+t"	Mở một tệp mới để đọc và ghi trong chế độ văn bản, nếu tệp đã tồn tại nội dung của nó sẽ bị xoá.
"a+" ; "a+t"	Mở một tệp để đọc và ghi bổ sung trong chế độ văn bản, nếu tệp chưa tồn tại nó sẽ được tạo mới.
"r+b"	Mở một tệp để đọc và ghi trong chế độ nhị phân, sẽ có lỗi nếu tệp không tồn tại.
"w+b"	Mở một tệp mới để đọc và ghi trong chế độ nhị phân, nếu tệp đã tồn tại nội dung của nó sẽ bị xoá.
"a+b"	Mở một tệp để đọc và ghi bổ sung trong chế độ nhị phân, nếu tệp chưa tồn tại nó sẽ được tạo mới.

Công dụng: Hàm dùng để mở tệp theo các chế độ chỉ định. Nếu thành công hàm trả về một con trỏ kiểu *FILE* (*con trỏ tệp*) ứng với tệp vừa được mở. Nếu có lỗi, hàm trả về giá trị *NULL*.



Hình 5.2. Tiến trình mở tệp tin trong ngôn ngữ lập trình C

Ví dụ 5-1. Để mở một tệp có tên là ***VanBan.txt*** do con trỏ tệp ***PT*** trỏ tới dùng để ghi theo kiểu văn bản và một tệp có tên là ***NhiPhan.bir*** do con trỏ tệp ***PB*** trỏ tới dùng để ghi theo kiểu nhị phân ta làm như sau:

```
...
FILE *PT, *PB; /* Khai báo các con trỏ tệp */
```

```
PT=fopen("VanBan.txt", "wt");
PB=fopen("NhiPhan.bir", "wb");
...
```

Chú ý:

- Một tệp tin sau khi đã mở ta mới có thể thực hiện các thao tác đọc/ghi trên đó bằng các hàm đọc ghi.

- Trong khi viết chương trình luôn phải kiểm tra xem liệu thao tác mở tệp tin có thành công hay không? Nếu thành công mới tiến hành các thao tác tiếp theo, ngược lại phải thông báo tình hình lỗi cho người sử dụng biết. Để thực hiện điều đó thông thường ta dùng đoạn chương trình sau:

Ví dụ 5-2. Đoạn chương trình dùng kiểm tra lỗi khi mở tệp:

```
...
char Ten[20];
FILE *f; /*Khai báo con trỏ tệp*/
printf("\nNhap ten tep nguon"); gets(Ten);
f= fopen(Ten, "rb"); /* Mở tệp tin để đọc trong chế độ nhị phân */
if(f==NULL) /*Nếu không mở được tệp*/
{
        printf("\nKhong mo duoc tep %s",Ten);
        getch(); /*Dừng chương trình để xem thông báo */
        exit(-1); /*Thoát*/
}
...
```

- Trong các kiểu làm việc ở cả hai chế độ ***vừa đọc vừa ghi*** thì ta cần phải có thao tác làm sạch vùng đệm bằng hàm ***fflush*** trước khi chuyển từ đọc sang ghi hoặc ngược lại để tránh mất mát dữ liệu theo cú pháp như sau:

```
int fflush(FILE* PF);
```

Công dụng: Hàm sẽ làm sạch vùng đệm của tệp đang hoạt động do con trỏ tệp ***PF*** trỏ đến. Nếu thành công hàm cho giá trị 0, ngược lại cho *EOF*.

Để có thể làm sạch tất cả các vùng đệm đang hoạt động ta dùng hàm thư viện:

```
int fflushall(void);
```

Nếu thành công hàm trả về số tệp đang mở, ngược lại cho *EOF*.

Các tệp đã mở sau khi kết thúc hoạt động cần được đóng lại để đảm bảo *không bị mất mát và hư hỏng thông tin*. Để đóng một tệp đang hoạt động do con trỏ tệp *PF* trỏ tới ta dùng hàm thư viện sau:

```
int fclose(FILE* PF);
```

Công dụng

Hàm dùng để đóng tệp. Thao tác đóng tệp là một chuỗi các thao tác dưới đây:

- *Đẩy dữ liệu còn trong vùng đệm lên đĩa (khi đang ghi).*
- *Xóa vùng đệm (khi đang đọc).*
- *Giải phóng con trỏ tệp để có thể dùng cho các tệp khác.*

Nếu thành công hàm cho giá trị *0*, ngược lại cho *EOF*.

Để có thể đóng hết các tệp đang mở ta dùng hàm sau:

int fcloseall(void);

Nếu thành công hàm trả về giá trị nguyên bằng số tệp đóng được, ngược lại hàm trả về *EOF*.

Sau khi tệp tin đã mở xong mà không có lỗi, ta có thể thao tác trên tệp tin đó theo chế độ đã chọn. Tuy nhiên, trong khi thực hiện các thao tác *đọc/ghi* vẫn có thể phát sinh các lỗi ổ đĩa, lỗi thiết bị phần cứng:.. Để có thể kiểm tra và thông báo được các lỗi dạng này trong quá trình thao tác trên tệp ta dùng hàm thư viện sau đây:

int ferror(FILE* PF);

Công dụng: Hàm cho giá trị *0* nếu không có lỗi xảy ra khi đang thao tác trên tệp do con trỏ tệp *PF* trỏ tới, trái lại hàm cho một giá trị khác không.

Hàm này thường được dùng kết hợp với hàm *perror* để chỉ ra nội dung của lỗi đã phát ra trong lúc hoạt động:

void perror(const char *s);

Trong đó s là một chuỗi kí tự do người lập trình đưa vào để thông báo khi có lỗi.

Công dụng: Hàm đưa chuỗi *s* và một thông báo lỗi của hệ thống tương ứng với lỗi đã phát sinh ra màn hình.

Ví dụ 5-3. Đoạn chương trình dùng để thông báo lỗi trong quá trình đọc/ghi trên tệp.

```
...
int c;
char Ten[20];
FILE *f; /*Khai báo con trỏ tệp*/
printf("\nNhap ten tep nguon"); gets(Ten);
f=fopen(Ten, "rb"); /*Mở tệp để đọc theo kiểu nhị phân */
if(f==NULL) /*Nếu không mở được tệp*/
{
        printf("\nKhong mo duoc tep %s",Ten);
        getch(); /*Dừng chương trình để xem dòng thông báo*/
        exit(-1); /*Thoát*/
}
c=fgetc(f); /*Đọc một kí tự vào c*/
if(ferror(f)) /*Nếu có lỗi*/
{
        perror("Loi doc tep : ");
        fclose(f);
        exit(-1);
}
...
```

5.3.2. Xuất/nhập kí tự

Mỗi hàm trong thư viện các hàm cấp 2 dùng để truy xuất tệp đều có thể làm việc trong cả hai chế độ (*văn bản và nhị phân*) tuy nhiên để tránh các sai sót có thể xảy ra ta nên phân biệt các hàm thường dùng trong chế độ văn bản và các hàm

thường dùng trong chế độ nhị phân. Trong mục này ta sẽ nghiên cứu các hàm nên làm việc trong chế độ văn bản.

1. Ghi kí tự lên tệp dùng hàm putc và fputc

Dạng hàm: int putc(int ch, FILE*PF);
int fputc(int ch, FILE*PF);

Trong đó, ch là một giá trị nguyên không dấu, *PF* là con trỏ trỏ đến tệp đã được mở bằng hàm ***fopen*** theo kiểu *ghi* hoặc *đọc/ghi*.

Công dụng: Hàm sẽ ghi lên tệp do con trỏ tệp *PF* trỏ tới tại vị trí của *con trỏ chỉ vị* một kí tự có mã bằng *ch%256*. Nếu thành công, hàm trả về mã của kí tự được ghi, ngược lại hàm cho *EOF*.

Chú ý: Hai hàm trên hoạt động tương tự nhau.

Ví dụ 5.4. Viết chương trình nhập từ bàn phím một chuỗi kí tự, các kí tự này sẽ lần lượt được ghi vào tệp văn bản cho đến khi gõ *ESC* (*mã bằng 27*). Chương trình có kiểm tra các trường hợp lỗi và thông báo ra màn hình. Tên tệp nhập từ bàn phím.

```
/* ****************************************************************************** */
#include "stdio.h"
#include "conio.h"
#include "process.h"
/* ****************************************************************************** */
int main()
{
    char ch;
    char Ten[20];
    FILE *F; /*Khai báo con trỏ tệp*/
    printf("\nNhap ten tep nguon"); gets(Ten);
    F=fopen(Ten, "w"); /* Mở một tệp mới để ghi trong chế độ văn bản */
    if(F==NULL) /*Nếu không mở được tệp*/
    {
        printf("\nKhong mo duoc tep %s",Ten);
        getch(); /*Dừng chương trình để xem*/
        exit(-1); /*Thoát, hàm này được khai báo trong thư viện process.h */
    }
    /*Gõ kí tự nào, ghi nó luôn vào tệp đã mở */
    while((ch=getche())!=27)
    {
        putc(ch, F); /* Ghi kí tự vừa gõ vào tệp do con trỏ F trỏ tới */
        if(ferror(F)) /*Kiểm tra lỗi*/
        {
            perror("Loi ghi tep : ");
            fclose(F);
            exit(-1);
        }
        if(ch=='\r') printf("\n");
    }
    fclose(F);
    getch();
    return 0;
}
```

__2. Đọc kí tự từ tệp dùng hàm getc và fgetc__

Dạng hàm:

int getc(FILE*f);
int fgetc(FILE*f);

Trong đó *f* là con trỏ tệp trỏ đến tệp mở để *đọc* hoặc *đọc/ghi* theo kiểu văn bản.

Công dụng: Cả hai hàm đều thực hiện đọc một kí tự từ tệp do con trỏ *f* quản lí. Nếu thành công chúng cho mã kí tự đọc được (*từ 0 đến 255*), nếu gặp cuối tệp hay có lỗi chúng trả về *EOF*.

Chú ý:

Khi đọc dữ liệu từ tệp bao giờ ta cũng phải để ý đến điều kiện kết thúc tệp (*dấu hiệu EOF*), sau đây là một ví dụ minh họa cho điều đó.

Ví dụ 5.5. Viết chương trình hiện lên màn hình nội dung của một tệp văn bản. Trên mỗi dòng đặt *10* kí tự. Mỗi kí tự sẽ được thể hiện bởi hai thông số là mã và dạng kí tự. Sau khi cho hiện *24* dòng, chương trình tạm dừng cho đến khi bấm phím bất kì để xem tiếp.

```
/* ******************************************************************************** */
#include "stdio.h"
#include "conio.h"
#include "process.h"
/* ******************************************************************************** */
int main()
{
    char ch;
    int i,k;
    char Ten[20];
    FILE *F; /*Khai báo con trỏ tệp*/
    printf("\nNhap ten tep nguon"); gets(Ten);
    F=fopen(Ten, "r"); /* Mở để đọc trong chế độ văn bản */
    if(F==NULL) /*Nếu không mở được tệp*/
    {
        printf("\nKhong mo duoc tep %s",Ten);
        getch(); /*Dừng chương trình để xem*/
        exit(-1); /*Thoát*/
    }
    i=0; k=0; clrscr();
    /*Đọc kí tự từ tệp cho đến cuối tệp*/
    while((ch=fgetc(F))!=EOF)
    {
        /*Nếu có lỗi*/
        if(ferror(F)) /*Kiểm tra lỗi*/
        {
            perror("Loi ghi tep : ");
            fclose(F);
            exit(-1);
        }
        /*In mã của kí tự và bản thân kí tự*/
```

```
        printf("%5d: %c", ch, ch);
        ++i; /*Đếm số kí tự đã hiện trên một dòng*/
        if(i%10==0) /*Mỗi dòng 10 kí tự*/
        {
            ++k; /*Tăng số dòng*/
            printf("\n"); /*Sang dòng mới*/
        }
        if(k%24==0) /* Mỗi trang 24 dòng*/
        {
            getch();/*Dừng từng trang màn hình*/
            clrscr();/*Bắt đầu một trang mới*/
            k=1 ;
        }
    }
    fclose(F);
    getch();
    return 0;
}
/* ****************************************************************************** */
```

5.3.3. Xuất/nhập văn bản

1. Ghi dữ liệu lên tệp theo khuôn dạng dùng hàm fprintf

Dạng hàm:

int fprintf(FILE*f, const *DK, ...);

Trong đó, *f* là con trỏ tệp trỏ đến tệp cần ghi. DK chứa địa chỉ của chuỗi điều khiển, dấu '...' để chỉ danh sách các giá trị cần ghi lên tệp. Cách làm việc của hàm này về nguyên tắc giống hệt cách làm việc của hàm ***printf*** mà ta đã quen thuộc, chỉ khác một điều là hàm ***printf*** sẽ xuất dữ liệu theo khuôn dạng ra màn hình, còn hàm ***fprintf*** sẽ xuất dữ liệu theo khuôn dạng ra tệp do con trỏ *f* quản lí. Hàm này chỉ nên làm việc theo chế độ văn bản.

Ví dụ 5-6. Chương trình dưới đây sẽ tạo ra một tệp văn bản có nội dung như sau:

```
Danh sach cac hoc sinh gioi:
1. Nguyen Van A
2. Nguyen Van B
3. Nguyen Van C
```

```
/* ****************************************************************************** */
#include "stdio.h"
#include "conio.h"
#include "process.h"
/* ****************************************************************************** */
int main()
{
    int i;
    FILE *F; /*Khai báo con trỏ tệp*/
    F=fopen("DanhSach.Dat", "wt");
    if(F==NULL) /*Nếu không mở được tệp*/
```

```
    {
        printf("\nKhong mo duoc tep nay");
        getch(); /*Dừng chương trình để xem*/
        exit(-1); /*Thoát*/
    }
    /*Ghi dữ liệu ra tệp F*/
    fprintf(F, "Danh sach cac hoc sinh gioi:");
    for(i=1; i<=3; ++i)
        fprintf(F, "\n%d. Nguyen Van %c", i, 64+i);
    fclose(F);
    getch();
    return 0;
}
/* ****************************************************************************** */
```

2. Đọc dữ liệu từ tệp theo khuôn dạng dùng hàm fscanf

Dạng hàm:

int fscanf(FILE*f, const *DK, ...);

Trong đó, *f* là con trỏ tệp trỏ đến tệp cần đọc. *DK* chứa địa chỉ của chuỗi điều khiển, dấu '...' để chỉ danh sách các đối số chứa kết quả đọc được từ tệp. Cách làm việc của hàm này về nguyên tắc giống hệt cách làm việc của hàm ***scanf*** mà ta đã quen thuộc, chỉ khác một điều là hàm ***scanf*** sẽ nhập dữ liệu theo khuôn dạng từ bàn phím, còn hàm ***fscanf*** sẽ lấy dữ liệu theo khuôn dạng từ tệp do con trỏ ***f*** quản lí. Hàm này chỉ nên làm việc theo chế độ văn bản.

Ví dụ 5-7. Giả sử có một dãy số nguyên ghi trên tệp *SoLieu.DAT*, giữa hai số nguyên có ít nhất một khoảng trống hay dấu xuống dòng. Hãy viết chương trình đọc và in ra màn hình dãy số nói trên.

Cần phân biệt hai trường hợp:

- Sau chữ số cuối cùng là mã 26 hay cuối tệp.

- Sau chữ số cuối cùng có ít nhất một khoảng trống hay các dấu xuống dòng. ***Trường hợp này được coi như một bài tập.***

```
/* ****************************************************************************** */
/*Chương trình chỉ viết cho trường hợp thứ nhất . Chương trình sử dụng hàm int feof(FILE*f) để
kiểm tra cuối tệp, hàm cho giá trị khác không nếu gặp cuối tệp, ngược lại cho giá trị 0*/
#include "stdio.h"
#include "conio.h"
#include "process.h"
/* ****************************************************************************** */
int main()
{
    int c;
    FILE *F; /*Khai báo con trỏ tệp*/
    F=fopen("SoLieu.Dat", "r");
    if(F==NULL) /*Nếu không mở được tệp*/
    {
        printf("\nKhong mo duoc tep nay");
```

```
        getch(); /*Dừng chương trình để xem*/
        exit(-1); /*Thoát*/
    }
    /*Đọc cho đến cuối tệp*/
    while(!feof(F))
    {
        fscanf(F, "%d", &c);
        printf("\n%d", c);
    }
    fclose(F);
    getch();
    return 0;
}
/* ******************************************************************************** */
```

3. Ghi một chuỗi kí tự lên tệp dùng hàm fputs

Dạng hàm:

int fputs(const char *s, FILE*f);

Hàm sẽ ghi chuỗi kí tự đặt trong s lên tệp *f* (*dấu kết thúc chuỗi không được ghi vào tệp*), khi thành công hàm trả về kí tự cuối cùng được ghi lên tệp. Ngược lại trả về *EOF*.

Bài tập: Hãy viết chương trình nhập vào các dòng kí tự từ bàn phím sau đó ghi lên tệp có tên *VanBan.txt*.

4. Đọc một chuỗi kí tự từ tệp dùng hàm fgets

Dạng hàm:

char *fgets(char*s, int n, FILE *f);

Trong đó *s* là một con trỏ kiểu kí tự trỏ đến một vùng nhớ đủ lớn để chứa chuỗi kí tự đọc từ tệp, *n* là số nguyên xác định chiều dài tối đa của dãy cần đọc.

Công dụng: Hàm sẽ tiến hành đọc một dãy kí tự từ tệp do con trỏ *f* trỏ tới chứa vào vùng nhớ *s*. Việc đọc sẽ kết thúc khi:

- Hoặc đã đọc được n-1 kí tự.

- Hoặc gặp dấu xuống dòng (cặp giá trị 13 và 10), khi đó mã 10 được đưa vào xâu kết quả.

- Hoặc kết thúc tệp.

Xâu kết quả sẽ tự động được bổ sung dấu hiệu kết thúc chuỗi '\0'. Khi thành công hàm trả về địa chỉ vùng nhận kết quả, ngược lại cho giá trị *NULL*.

Bài tập: Hãy viết chương trình đọc dữ liệu từ tệp SoLieu.DAT đã được tạo ra trong ví dụ trước. Đưa kết quả ra màn hình.

Chú ý:

Trong quá trình làm việc Turbo C sẽ tự động mở sẵn các tệp tin và con trỏ tệp tin ứng với các thiết bị chuẩn cho trong bảng dưới đây. Trong chương trình, ta có

thể dùng các con trỏ này để nhập /xuất trên các thiết bị đó thông qua các tệp tin tương ứng.

Bảng 5.2. Các tệp tin và con trỏ tệp tin ứng với các thiết bị chuẩn

Tên tệp tin	Con trỏ tệp tin	Thiết bị tương ứng
in	stdin	Thiết bị vào chuẩn (bàn phím)
out	stdout	Thiết bị ra chuẩn (màn hình)
err	stderr	Thiết bị lỗi chuẩn (màn hình)
prn	stdprn	Thiết bị in chuẩn (máy in)

Ví dụ 5-8. Đoạn chương trình minh họa việc nhập/xuất các thiết bị chuẩn thông qua các tệp tin chuẩn tương ứng:

```
...
char HoTen[30];
float Diem;
printf("\nNhap ho ten tu ban phim :");
fgets(HoTen, 30, stdin); /*Đọc một chuỗi từ thiết bị vào chuẩn – Bàn phím*/
printf("\nNhap diem");
fscanf(stdin, "%f", &Diem); /*Đọc một số thực theo khuôn dạng từ thiết bị vào
chuẩn – Bàn phím */
printf("\nKet qua thi cua sinh vien %s la:", HoTen);
fprintf(stdout,"%f", Diem);
...
```

Việc kết nối giữa các tệp tin chuẩn ở trên có thể được định hướng lại cho việc gắn đến các thiết bị chuẩn. Việc định hướng này được thực hiện khi làm việc với các hàm cấp 1 bằng cách gắn trực tiếp việc xuất/nhập với các số hiệu tệp tin mặc định mà không cần phải mở chúng. Ví dụ:

- *Số hiệu 0 là tệp gắn với bàn phím (stdin);*
- *số hiệu 1 là tệp gắn với màn hình (stdout);*
- *số hiệu 2 là tệp gắn với màn hình (stderr);*
- *số hiệu 3 là tệp gắn với cổng nối tiếp và*
- *số hiệu 4 là tệp gắn với máy in (stdprn).*

Trong giáo trình này, chúng tôi không giới thiệu cách thức làm việc với tệp tin theo các hàm cấp 1.

5.3.4. Xuất/nhập nhị phân

1. Ghi một số nguyên lên tệp dùng hàm putw

Dạng hàm:

int putw(int n, FILE*f);

Hàm sẽ thực hiện ghi số nguyên ***n*** lên tệp *f* dưới dạng 2 byte. Nếu thành công hàm trả về số nguyên ghi được, ngược lại hàm trả về *EOF*.

2. Đọc một số nguyên từ tệp dùng hàm getw

Dạng hàm:

int getw(FILE*f);

Hàm sẽ thực hiện đọc một số nguyên từ tệp *f* dưới dạng 2 byte. Nếu thành công hàm trả về số nguyên đọc được, ngược lại hàm trả về *EOF*.

Ví dụ 5-9. Hãy viết chương trình thực hiện việc sao dữ liệu từ một tệp chứa các số nguyên có tên *SoNguyen.DAT* sang tệp khác có tên *Luu.Dat*.

```
/* ******************************************************************************** */
#include "stdio.h"
#include "conio.h"
#include "process.h"
/* ******************************************************************************** */
int main()
{
    int n;
    FILE *F1, *F2; /*Khai báo các con trỏ tệp*/
    F1=fopen("SoNguyen.Dat", "rb");/* Mở để đọc tệp nguồn*/
    if(F1==NULL)/*Nếu không mở được tệp*/
    {
        printf("\nKhong mo duoc tep nay");
        getch(); /*Dừng chương trình để xem*/
        exit(-1); /*Thoát*/
    }
    F2=fopen("Luu.DAT", "wb"); /*Mở tệp F2 để ghi dữ liệu*/
    if(F2==NULL) /*Nếu không mở được tệp*/
    {
        printf("\nKhong mo duoc tep nay");
        getch(); /*Dừng chương trình để xem*/
        exit(-1); /*Thoát*/
    }
    /*Đọc cho đến cuối tệp F1*/
    while((n=getw(F1))!=EOF)
        putw(n, F2); /*Ghi dữ liệu vừa đọc vào F2*/
    fclose(F1);
    fclose(F2);
    getch();
    return 0;
}
/* ******************************************************************************** */
```

3. Ghi một bản tin (cấu trúc) lên tệp dùng hàm fwrite

Dạng hàm:

`int fwrite(void *ptr, int size, int n, FILE*f);`

Hàm sẽ thực hiện ghi ***n*** bản tin kích thước ***size*** bytes từ vùng nhớ do ***ptr*** trỏ tới lên tệp *f*. Hàm trả về số bản tin thực sự ghi được.

4. Đọc một bản tin từ tệp dùng hàm fread

Dạng hàm:

`int fread(void *ptr, int size, int n, FILE*f);`

Hàm sẽ thực hiện đọc ***n*** bản tin kích thước ***size*** bytes từ tệp *f* chứa vào vùng nhớ do ***ptr*** trỏ tới. Hàm trả về số bản tin thực sự đọc được.

Ví dụ 5-10. Hãy viết chương trình nhập từ bàn phím một lượng các thông tin về nhân sự (*số lượng các bản tin chưa biết trước*), ghi vào tệp sau đó đọc tệp đó và hiện kết quả ra màn hình.

```
/* ****************************************************************************************** */
#include "stdio.h"
#include "conio.h"
#include "process.h"
typedef struct
{
    char HoTen[30];
    char ChucVu[20];
    int NamSinh;
} NhanSu;
/* ****************************************************************************************** */
int main()
{
    NhanSu NSu;
    char Ten[20];
    FILE *F; /*Khai báo con trỏ tệp*/
    printf("\nNhap ten tep nguon"); gets(Ten);
    F= fopen(Ten, "wb"); /* Mở để ghi mới trong chế độ nhị phân */
    if(F==NULL)/*Nếu không mở được tệp*/
    {
        printf("\nKhong mo duoc tep %s",Ten);
        getch(); /*Dừng chương trình để xem*/
        exit(-1); /*Thoát*/
    }
    /*Nhập số liệu từ bàn phím rồi ghi lên tệp*/
    while(1) /* Lặp không điều kiện cho đến khi gõ Enter khi nhập tên */
    {
        printf("\nNhap ho va ten(Bam Enter de ket thuc)"); gets(NSu.HoTen);
        if(NSu.HoTen[0]=='\0') break; /*Thoát nếu không nhập tên*/
        printf("\nChuc vu cua Ong(Ba) %s la: ", NSu.HoTen); gets(NSu.ChucVu);
        printf("\nOng(Ba) %s sinh nam : ", NSu.HoTen);
        scanf("%d%*c", &NSu.NamSinh);
        fwrite(&NSu, sizeof(NhanSu), 1, F); /*Ghi bản tin ra tệp F */
        if(ferror(F))/*Kiểm tra lỗi*/
        {
            perror("Loi ghi tep: ");
            fclose(F);
            exit(-1);
        }
    }
    fclose(F); /*Đóng tệp trước khi đọc*/
    /*Đọc dữ liệu từ tệp rồi đưa ra màn hình*/
    F=fopen(Ten, "rb"); /*Mở để đọc*/
    while(fread(&NSu, sizeof(NhanSu),1, F)>0)
        printf("\nOng(Ba) %s giu vi tri %s sinh nam %d", NSu.HoTen,
        NSu.ChucVu, NSu.NamSinh);
    fclose(F);
```

```
    getch();
    return 0;
}
/* ******************************************************************************************* */
```

5.3.5. Xuất/nhập ngẫu nhiên

1. Chuyển con trỏ chỉ vị về đầu tệp

Dạng hàm:

void rewind(FILE*F);

Hàm thực hiện chuyển con trỏ chỉ vị của tệp do *F* trỏ đến về đầu tệp.

2. Chuyển con trỏ chỉ vị đến vị trí bất kì

Dạng hàm:

int fseek(FILE*f, long SoByte, int XuatPhat);

Hàm sẽ thực hiện di chuyển con trỏ chỉ vị của tệp do con trỏ *f* trỏ đến từ vị trí ban đầu được xác định bởi tham số ***XuatPhat*** qua một số byte đúng bằng | ***SoByte***|, chiều dịch chuyển sẽ tiến về cuối tệp nếu giá trị của *SoByte* là dương, ngược lại sẽ tiến về đầu tệp. Nếu thành công hàm trả về *0*, ngược lại sẽ trả về một số khác không. Giá trị của tham số *XuatPhat* có thể nhận các giá trị sau đây:

- *SEEK_SET hay 0: Xuất phát từ đầu tệp.*
- *SEEK_CUR hay 1: Xuất phát từ vị trí hiện tại của con trỏ chỉ vị.*
- *SEEK_END hay 2: Xuất phát từ cuối tệp.*

Chú ý:

fseek chỉ nên dùng trong chế độ nhị phân do sự chuyển đổi các kí tự đặc biệt trong chế độ văn bản có thể dẫn đến việc định vị sai vị trí cần đến.

3. Tìm vị trí hiện tại của con trỏ chỉ vị

Dạng hàm:

long ftell(FILE*f);

Khi thành công, hàm cho biết vị trí hiện tại của con trỏ chỉ vị (*nằm ở byte thứ bao nhiêu bắt đầu từ vị trí 0*), ngược lại hàm cho -1L.

Ví dụ 5-11. Hãy viết chương trình xác định kích thước của một tệp bất kì.

```
/* ******************************************************************************************* */
#include "stdio.h"
#include "conio.h"
#include "process.h"
/* ******************************************************************************************* */
int main()
{
    char Ten[20];
    long n;
    FILE *F; /*Khai báo con trỏ tệp*/
    printf("\nNhap ten tep nguon"); gets(Ten);
```

```
        F=fopen(Ten, "rb");
        if(F==NULL)/*Nếu không mở được tệp*/
        {
                printf("\nKhong mo duoc tep %s",Ten);
                getch(); /*Dừng chương trình để xem*/
                exit(-1); /*Thoát*/
        }
        /*Tính độ dài tệp*/
        fseek(F, 0, SEEK_END); /*Chuyển về cuối tệp*/
        n=ftell(F);
        fclose(F);
        printf("\nĐo dai tep %s là %ld byte",Ten, n);
        getch();
        return 0;
}
/* ********************************************************************************************* */
```

5.4. CÂU HỎI VÀ BÀI TẬP

1. Vai trò của con trỏ tệp tin là gì? phân biệt nó với con trỏ chỉ vị? Tại một thời điểm mỗi tệp có mấy con trỏ tệp? mấy con trỏ chỉ vị? Ta có thể tạo ra con trỏ chỉ vị được không? Tại sao?
2. Phân biệt xuất / nhập nhị phân với xuất nhập văn bản? Tại sao ta phải sử dụng hai kiểu làm việc này ?
3. Phân biệt truy xuất tuần tự và truy xuất ngẫu nhiên? Đặc điểm của từng loại ?
4. Vai trò của việc mở, đóng tệp tin là gì? Có nhất thiết phải thực hiện trong khi thao tác với tệp hay không? Tại sao?
5*. Các câu lệnh sau đây sẽ ghi gì lên tệp do con trỏ *F* trỏ tới, nếu tệp đó đang làm việc theo chế độ văn bản:
putc(-1,F); fputc(10,F); putc(26,F); putc(1820, F); putc(5,F); putc(26,F);
Ta sẽ nhận được những gì nếu ta dùng lệnh *type* của DOS để xem tệp tin vừa ghi? Khi mở tệp này theo chế độ văn bản và theo chế độ nhị phân ta sẽ có kết quả như thế nào? Tại sao?
6. Phân biệt *EOF* và mã *26* ? Giải thích?
7*. Nếu ta thực hiện việc sao từ tệp *f1* sang tệp *f2* trong chế độ văn bản theo thuật toán sau:

```
while(!feof(f1))
        fput(fgetc(f1), f2);
```

thì sẽ thu được kết quả như thế nào? Hãy so sánh hai tệp *f1* và *f2* với nhau! Hãy giải thích tại sao có kết quả như vậy!
8. Nếu một cấu trúc có chứa thành phần con trỏ trong đó có thể ghi được ra tệp hay không ? Nếu ghi được thì phải làm gì để không mất mát thông tin? Viết lại chương trình ví dụ *5-10* nếu dữ liệu được tổ chức theo kiểu danh sách móc nối.
9*. Viết chương trình sửa nội dung của một tệp bất kì (*tên tệp nhập từ bàn phím*). Chương trình cho hiện lên màn hình từng trang 200 kí tự của tệp. Mỗi kí tự sẽ được hiện dưới dạng mã *ASCII* và dạng kí tự (*nếu có*) của nó. Có thể dùng các phím mũi tên để di chuyển đến mã cần sửa. Bấm Enter để xem/sửa trang tiếp theo. Bấm *ESC* để kết thúc chương trình.
10. Viết chương trình đọc một dãy số từ bàn phím rồi ghi chúng vào đĩa mềm cho đến khi gặp số 0. Đọc từ đĩa, đưa ra dãy số đã ghi và tổng của chúng.

11. Lập một chương trình thực hiện các công việc sau:

a) Nhập từ bàn phím một danh sách sinh viên gồm họ tên, giới tính, năm sinh. Kết thúc nhập khi gặp họ tên = '*'.

b) Ghi dữ liệu ra đĩa mềm với tên tệp là SVLOP_X.

c) Đọc từ tệp SVLOP_X, tìm các sinh viên là nữ và sinh trước 1979, đưa kết quả ra màn hình.

12. Lập chương trình thực hiện các việc sau:

a) Đọc từ bàn phím một dãy n số nguyên.

b) Ghi dãy số đó và đĩa mềm.

c) Sắp xếp các số lẻ lên đầu dãy, các số chẵn xuống cuối dãy mà không được sử dụng thêm mảng mới.

Đưa ra màn hình dãy số đã sắp xếp, số lượng các số lẻ và tổng của chúng.

13. Có một danh sách hàng hóa gồm tên hàng, số lượng, đơn giá, thành tiền. Lập trình các việc sau:

a) Nhập dữ liệu từ bàn phím cho đến khi gặp số lượng bằng 0. Ghi danh sách vào tệp với tên tệp đọc từ bàn phím.

b) Thêm vào cuối danh sách một hàng hóa mới (tên, số lượng, đơn giá, thành tiền), ghi tiếp vào đĩa.

c) Đọc dữ liệu từ tệp đã ghi, trích tên hàng và số lượng ra một tệp riêng, tên tệp là TRICH.DAT.

d) Đọc tệp TRICH.DAT, đưa danh sách này ra màn hình.

14. Viết chương trình thực hiện các việc sau:

a) Tạo hai tệp F1, F2 là những tệp văn bản để ghi dữ liệu từ bàn phím.

b) Nối tệp F2 vào cuối tệp F1.

c) Đưa nội dung tệp F1 ra màn hình.

15. Có n mặt hàng (n < 50), mỗi mặt hàng gồm tên (không quá 10 kí tự), số lượng (số nguyên) và đơn giá (số thực). (Đơn giá của một mặt hàng là giá một đơn vị sản phẩm của mặt hàng đó).

a) Lập chương trình nhập các thông tin từ bàn phím và cất vào một tệp trên đĩa kiểu record với tên là DULIEU.

b) Lập một chương trình thực hiện các công việc sau:

+) Đọc tệp DULIEU và viết lên màn hình tất cả tên mặt hàng chứa trong tệp đang xét.

+) Vào từ bàn phím tên mặt hàng và số lượng cần xuất hay nhập. Chương trình sẽ tự động cập nhật số lượng mới của mặt hàng đang xét vào tệp DULIEU. Nếu vào không đúng tên mặt hàng, chương trình sẽ thông báo lỗi và đòi hỏi vào lại.

+) Tìm trong tệp DULIEU và ghi lên một tệp Text trên đĩa với tên BAOCAO.TXT tất cả những mặt hàng (gồm tên, số lượng, tổng giá trị) thỏa mãn tổng giá trị lớn hơn hay bằng một giá trị cho trước từ bàn phím. (Tổng giá trị của một mặt hàng bằng đơn giá nhân với số lượng của mặt hàng đó).

16. Hãy lập chương trình làm các việc sau:

a) Tạo một tệp, mỗi bản ghi gồm:

- Tên sản phẩm: xâu < 21 kí tự.
- Mã sản phẩm: gồm 2 mã, mỗi mã là một số nguyên.
- Đơn giá: số thực.

Tên tệp đọc từ bàn phím và kết thúc vào dữ liệu khi gặp tên rỗng.

b) Cập nhật đơn giá sản phẩm của các bản ghi trên tệp dựa theo mã thứ 2. Yêu cầu hiển thị lần lượt các bản ghi thuộc diện sửa chữa và cho phép người dùng sửa hay không sửa bản ghi tương ứng. Nếu gặp mã mới thì thông báo và bỏ qua.

Chương 6

ĐỒ HOẠ (GRAPHIC)

MỤC TIÊU CỦA CHƯƠNG NÀY

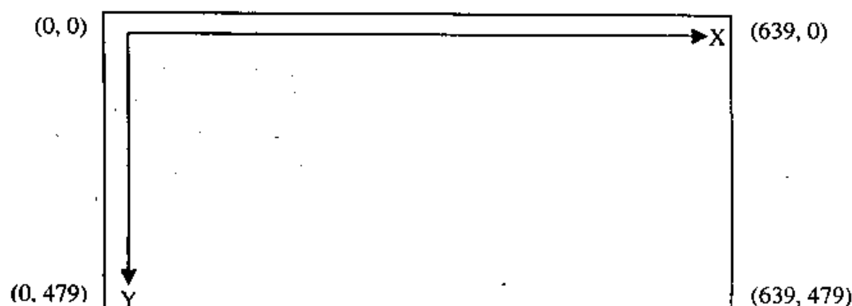
- *Hiểu các đặc điểm khi làm việc trong chế độ văn bản và chế độ đồ hoạ.*
- *Thực hiện thành thạo các thao tác cơ bản trong chế độ đồ hoạ.*
- *Có khả năng viết các chương trình ứng dụng làm việc trong chế độ đồ hoạ.*

6.1. MÀN HÌNH ĐỒ HOẠ VÀ CÁC ĐẶC TRƯNG

Trong hệ thống máy tính, thiết bị hiển thị bao gồm hai thành phần chính đó là *bộ tương hợp hiển thị* (*display adapter hay còn gọi là cạc màn hình*) điều khiển việc hiển thị dữ liệu (*phần này thường cắm trên bo mạch chủ hoặc được chế tạo gắn liền với bo mạch chủ - onboard*) và *màn hình hiển thị* (*display monitor*). Màn hình hiển thị (*là nơi mà các kí tự, hình vẽ... xuất hiện*) thực chất là tập hợp của các điểm sáng nhỏ (*gọi là các pixel – viết tắt của picture element*) và được bố trí thành một ma trận điểm gồm *m* dòng và *n* cột. Kích thước của ma trận điểm *nxm* này được gọi là độ phân giải (*resolution*) của màn hình hiển thị (*ví dụ màn hình độ phân giải 640 * 480 – sẽ có 480 dòng, mỗi dòng có 640 điểm ảnh*).

Máy tính có thể làm việc được trong hai chế độ hiển thị khác nhau, đó là *chế độ văn bản* và *chế độ đồ hoạ*. Từ trước tới nay chúng ta vẫn làm việc trong chế độ văn bản, đó là chế độ mà mọi dữ liệu được hiển thị ra theo các ô chữ nhật cố định (*gồm một số xác định các pixel như 8*8, 8*16, 9*16... tuỳ theo độ phân giải và kiểu hoạt động được thiết lập*) tạo thành một ma trận *m1* dòng, mỗi dòng gồm *n1* ô chữ nhật (*mỗi ô chữ nhật có thể hiển thị cho một kí tự*). Thông thường trong chế độ văn bản các *màn hình hiển thị* hiện nay có thể hiển thị được *25* dòng, mỗi dòng *80* kí tự.

Trong chế độ đồ hoạ ta có thể xử lí đến từng *pixel* và do đó về nguyên tắc có thể tạo ra một hình ảnh bất kì trên màn hình hiển thị. Tuỳ theo từng *bộ tương hợp hiển thị* được cài đặt trong máy và chế độ đồ hoạ được thiết lập mà ta có thể làm việc trong chế độ đồ hoạ với các độ phân giải tương ứng (*bảng 6.1. sẽ chỉ các thông số cơ bản khi làm việc trong chế độ đồ hoạ của một số bộ tương hợp thông dụng*). Để tăng cường khả năng tương thích và mềm dẻo khi làm việc với chế độ đồ hoạ, Turbo C cung cấp sẵn một số các *trình điều khiển màn hình* cho các loại *Adapter* khác nhau như CGA, VGA, EGA... dưới dạng các tệp tin có phần mở rộng là ***.bgi**; các tệp tin quy định *font* chữ trong chế độ đồ hoạ có phần mở rộng là ***.chr** và rất nhiều *hàm thư viện* phục vụ cho các thao tác trong chế độ đồ hoạ. Các hàm thư viện này được đặt trong tệp tin *graphics.lib* và được khai báo nguyên mẫu trong tệp tiêu đề *graphics.h* (*do đó ta cần ghép tệp tiêu đề này vào chương trình bằng chỉ thị #include nếu muốn làm việc trong chế độ đồ hoạ*). Màn hình hiển thị trong chế độ đồ hoạ (*hình 6.1*) là một ma trận điểm ảnh có gốc toạ độ (*0, 0*) tính từ góc cao bên trái của màn hình, mỗi điểm ảnh sẽ tương ứng với một toạ độ (*x,y*) trong hệ toạ độ đó. Mỗi khi ta cần truy nhập đến một điểm ảnh bất kì trên màn hình, ta sẽ truy nhập thông qua toạ độ của điểm ảnh trong hệ toạ độ của màn hình.



Hình 6.1. Hệ toạ độ trong chế độ đồ hoạ của màn hình VGA.

Bảng 6.1. Các thông số cơ bản của một số Adapter thông dụng và các chế độ đồ hoạ

Màn hình	Chế độ đồ hoạ	Độ phân giải	Bảng màu	Số trang
CGA [26] (1)	CGAC0 (0)	320 x 200	C0 (4 màu)	1
	CGAC1 (1)	320 x 200	C1 (4 màu)	1
	CGAC2 (2)	320 x 200	C2 (4 màu)	1
	CGAC3 (3)	320 x 200	C3 (4 màu)	1
	CGAHI (4)	640 x 200	2 màu	1
MCGA (2)	MCGAC0 (0)	320 x 200	C0 (4 màu)	1
	MCGAC1 (1)	320 x 200	C1 (4 màu)	1
	MCGAC2 (2)	320 x 200	C2 (4 màu)	1
	MCGAC3 (3)	320 x 200	C3 (4 màu)	1
	MCGAMED (4)	640 x 200	2 màu	1
	MCGAHI (5)	640 x 480	2 màu	1
EGA (3)	EGALO (0)	640 x 200	16 màu	4
	EGAHI (1)	640 x 350	16 màu	2
EGA64 (4)	EGA64LO (0)	640 x 200	16 màu	1
	EGA64HI (1)	640 x 350	4 màu	1
VGA (9)	VGALO (0)	640 x 200	16 màu	2
	VGAMED (1)	640 x 350	16 màu	2
	VGAHI (2)	640 x 480	16 màu	1
PC3270 (10)	PC3270HI (0)	720 x 350	2 màu	1
IBM8514 (6)	IBM8514LO (0)	640 x 480	256 màu	
	IBM8514HI (1)	1024 x 760	256màu	

6.2. KHỞI ĐỘNG CHẾ ĐỘ ĐỒ HOẠ

Muốn làm việc được trong chế độ đồ hoạ ta phải khởi động nó bằng hàm thư viện *initgraph()* theo cú pháp sau đây:

initgraph(&Driver, &Mode, Đường dẫn);

Trong đó *Driver* là một biến nguyên chứa giá trị tương ứng với trình điều khiển màn hình đang sử dụng (*ví dụ Driver = VGA hoặc Driver = 9 nếu ta đang sử dụng loại VGA*), *Mode* cũng là một biến nguyên dùng để chỉ định chế độ đồ hoạ muốn thiết lập (*giá trị này cũng cho trong bảng 6.1. Ví dụ với Driver = VGA thì ta có thể khởi tạo 3 chế độ làm việc tương ứng là VGALO, VGAMED và VGAHI*), còn *Đường dẫn* là một xâu kí tự chứa đường dẫn đến thư mục có chứa các tệp **.bgi* (*lưu ý đường dẫn này phải có dạng ví dụ như "C:\\LapTrinh\\TC2"*).

[26] Các tên này cũng chính là các tên hằng tương ứng với các trình điều khiển màn hình được định nghĩa trong graphics.h và giá trị của nó bằng giá trị nằm trong cặp ngoặc ().

Chú ý :

- Nếu ta khởi tạo chế độ đồ hoạ bằng các tham số không tương thích thì chương trình có thể sẽ không hoạt động. Do vậy, nếu ta không biết chính xác các tham số của thiết bị hiển thị thì ta nên dùng *Driver = DETECT* hoặc *Driver = 0*. Khi tham số *Driver* nhận các giá trị này chương trình sẽ *tự động phát hiện* các tham số tương ứng của thiết bị hiển thị với độ phân giải cao nhất có thể, sau đó gán cho *Driver* và *Mode* trước khi quá trình khởi động diễn ra.

- Nếu đường dẫn là một xâu rỗng "" thì tiến trình khởi động sẽ chỉ tìm kiếm các tệp **.bgi* trong thư mục chủ.

Nếu quá trình khởi động diễn ra *không thành công* thì hàm thư viện *graphresult* sẽ trả về một *mã lỗi* tương ứng với lỗi phát sinh, mã lỗi này có thể được nhận biết nhờ hàm *grapherrormsg*, hàm này sẽ trả về một con trỏ trỏ đến chuỗi thông báo tương ứng với mã lỗi đã phát sinh và ta có thể đưa ra màn hình (*xem ví dụ 6.1*).

Nếu quá trình khởi động diễn ra *thành công* (*hàm graphresult sẽ trả về giá trị grOk hoặc 0*) thì ta có thể thực hiện các thao tác có thể trong chế độ đồ hoạ (*thường sử dụng các hàm thư viện của Turbo C*). Để nhận biết độ phân giải của màn hình đồ hoạ vừa khởi tạo ta sử dụng các hàm thư viện *getmaxx* và *getmaxy*. Hàm *getmaxx()* sẽ trả về số điểm ảnh theo bề rộng màn hình đồ hoạ (*trục x*), còn hàm *getmaxy()* sẽ trả về số điểm ảnh theo bề cao của màn hình đồ hoạ (*trục y*). Sau khi thực hiện xong các thao tác trong chế độ đồ hoạ ta cần đóng chế độ đồ hoạ bằng hàm thư viện *closegraph* theo cú pháp sau : *closegraph() ;*

Ví dụ 6-1. Đoạn chương trình dùng để khởi động chế độ đồ hoạ.

```
...
int Driver = DETECT, Mode, MaLoi ;
initgraph(&Driver, &Mode, "") ;
Maloi = graphresult() ;
if(MaLoi) /* Nếu Mã lỗi khác 0 (có lỗi) */
{
      printf("\nMa loi la %d  ", MaLoi);
      puts(grapherrormsg(MaLoi)) ; /* Đưa ra thông báo tương ứng */
      exit(-1); /* Thoát */
}
...
```

6.3. CÁC THAO TÁC CƠ BẢN TRONG CHẾ ĐỘ ĐỒ HOẠ

6.3.1. Thiết lập màu nền và màu nét vẽ

Để thiết lập màu nền cho màn hình đồ hoạ ta sử dụng hàm thư viện theo cú pháp như sau: *setbkcolor(Mau);*

Còn để thiết lập màu cho nét vẽ bất kì ta dùng hàm: *setcolor(Mau);* Trong đó *Mau* là một tên hằng (*hoặc một giá trị nguyên tương ứng*) cho trong bảng 6.2. Ta cũng có thể định nghĩa lại bảng mầu này bằng các tên tiếng việt tương ứng bằng từ khoá *enum* cho tiện sử dụng như sau:

```
enum BangMau {DEN, XANH_DA_TROI, XANH_LA_CAY, XANH_LO, DO, TIM, NAU, XAM_NHAT, XAM_SAM, XANH_DA_TROI_NHAT, XANH_LA_CAY_NHAT, XANH_LO_NHAT, DO_NHAT, TIM_NHAT, VANG, TRANG};
```

Hãy ghi định nghĩa này vào tệp *graphics.h* (*cạnh với định nghĩa bảng màu bằng tiếng anh*) để khả dụng cho tất cả các chương trình ứng dụng mà bạn sẽ viết.

Bảng 6.2. Giải màu ngầm định được định nghĩa trong graphics.h

Tên hằng	Giá trị số	Màu hiển thị
BLACK	0	Màu đen.
BLUE	1	Màu xanh da trời.
GREEN	2	Màu xanh lá cây.
CYAN	3	Màu xanh lơ.
RED	4	Màu đỏ.
MAGENTA	5	Màu tím.
BROWN	6	Màu nâu.
LIGHTGRAY	7	Màu xám nhạt.
DARKGRAY	8	Màu xám sẫm.
LIGHTBLUE	9	Màu xanh da trời nhạt.
LIGHTGREEN	10	Màu xanh lá cây nhạt.
LIGHTCYAN	11	Màu xanh lơ nhạt.
LIGHTRED	12	Màu đỏ nhạt.
LIGHTMAGENTA	13	Màu tím nhạt.
YELLOW	14	Màu vàng.
WHITE	15	Màu trắng.

Để nhận lại mầu nền hiện tại ta dùng hàm : *getbkcolor()*;

Để nhận lại mầu nét vẽ hiện tại ta dùng hàm: *getcolor();*

6.3.2. Thiết lập kiểu, mẫu tô cho nét vẽ và hình vẽ

Để thiết lập được kiểu, mẫu tô và bề dày cho nét vẽ ta sử dụng hàm:

setlinestyle(Kiểu, Mẫu tô, Bề dày);

Trong đó tham số *Kiểu* có thể nhận một trong các giá trị sau :

Bảng 6.3. Các giá trị và tên hằng tương ứng cho các kiểu của nét vẽ

Tên hằng	Giá trị số	Mô tả
SOLID_LINE	0	Sẽ là nét vẽ đậm.
DOTTED_LINE	1	Sẽ là nét chấm chấm.
CENTER_LINE	2	Sẽ là nét chấm gạch.
DASHED_LINE	3	Sẽ là nét đứt.
USERBIT_LINE	4	Kiểu do người dùng tự định nghĩa.

Nếu tham số *Kiểu* có giá trị bằng 4 (*tự định nghĩa*) thì tham số *Mẫu tô* sẽ chỉ ra cách thức để xây dựng nét vẽ (*ví dụ nếu Mẫu tô = 0x 0101 thì nét vẽ sẽ có dạng như sau '. . . . . ', mỗi điểm chấm tương ứng với bit 1 trong Mẫu tô*).

Tham số *Bề dày* có thể nhận một trong hai giá trị là *1* (*NORM_WIDTH – nét vẽ dày 1 điểm ảnh*) hoặc *3* (*THICK_WIDTH – nét vẽ dày 3 điểm ảnh*).

Để thiết lập màu, kiểu, mẫu tô cho các hình vẽ ta dùng hàm:

setfillstyle(Mẫu tô, Màu tô);

Trong đó tham số *Màu tô* có thể nhận một trong các giá trị hoặc tên hằng cho trong bảng 6.2, còn tham số *Mẫu tô* có thể nhận một trong các giá trị hoặc tên hằng cho trong bảng 6.4 dưới đây.

Bảng 6.4. Các giá trị và tên hằng tương ứng cho các kiểu tô của hình vẽ

Tên hằng	Giá trị số	Mô tả
EMPTY_FILL	0	Màu nền.
SOLID_FILL	1	Tô kín bằng màu đã chỉ định.
LINE_FILL	2	Tô bằng các nét ngang ----
LTSLASH_FILL	3	Tô bằng các nét gạch chéo /// mảnh.
SLASH_FILL	4	Tô bằng các nét gạch chéo /// dày.
BKSLASH_FILL	5	Tô bằng các nét gạch chéo ngược \\\ dày.
LTBKSLASH_FILL	6	Tô bằng các nét gạch chéo ngược \\\ mảnh.
HATCH_FILL	7	Tô bằng nét sọc dừa thưa.
XHATCH_FILL	8	Tô bằng nét sọc dừa dày.
INTERLEAVE_FILL	9	Tô bằng các đường xen kẽ.
WIDE_DOT_FILL	10	Tô bằng các dấu chấm thưa.
CLOSE_DOT_FILL	11	Tô bằng các dấu chấm dày.

Hàm này sẽ có tác dụng đối với các hàm *bar*, *fillpoly*, *pieslice* và *floodfill*.

6.3.3. Vẽ đường tròn và hình tròn

1. Hàm arc

Dạng hàm: void arc(int x, int y, int GocDau, int GocCuoi, int r);

Công dụng: hàm vẽ ra một cung tròn trên màn hình đồ hoạ có tâm tại toạ độ (*x*,*y*), có bán kính là *r*, có góc ban đầu là *GocDau* và góc kết thúc cung là *GocCuoi* [27].

2. Hàm circle

Dạng hàm: void circle(int x, int y, int r);

Công dụng: hàm vẽ ra một đường tròn trên màn hình đồ hoạ có tâm tại toạ độ (*x*,*y*) và có bán kính là *r*.

3. Hàm ellipse

Dạng hàm: void ellipse(int x, int y, int GocDau, int GocCuoi, int xr, int yr);

Công dụng: hàm vẽ ra một cung ellip trên màn hình đồ hoạ có tâm tại toạ độ (*x*,*y*), có bán kính là trục ngang *xr*, có bán kính trục đứng *yr*, có góc ban đầu *GocDau* và có góc kết thúc *GocCuoi*.

4. Hàm pieslice

Dạng hàm: void pieslice(int x, int y, int GocDau, int GocCuoi, int r);

Công dụng: hàm vẽ và tô màu cho một hình quạt trên màn hình đồ hoạ có tâm tại toạ độ (*x*,*y*), có bán kính là *r*, có góc ban đầu là *GocDau* và có góc kết thúc là *GocCuoi* (*nếu GocDau=0 và GocCuoi=360 thì ta được một hình tròn có tô màu*).

Ví dụ 6-2. Viết chương trình vẽ một hình tròn tô màu xanh da trời ở chính giữa màn hình, nét vẽ của đường tròn màu đỏ, độ dày của nét vẽ là 3, màu nền trắng.

```
/* ************************************************************************************************ */
#include "graphics.h"
#include "conio.h"
#include "stdio.h"
```

[27] Các góc đều tính theo đơn vị là độ (từ 0 đến 360 độ)

```
#include "process.h"
/* ************************************************************************************************ */
int main()
{
	int Driver =DETECT, Mode, MaLoi;
	initgraph(&Driver, &Mode, "") ;
	MaLoi=graphresult() ;
	if(MaLoi)
	{
		printf(" \nLoi do hoa, ma loi la:  %d\n", MaLoi) ;
		puts(grapherrormsg(MaLoi)) ;
		exit(-1) ;
	}
	setbkcolor(WHITE); /* đặt nền màu trắng */
	setcolor(RED); /* nét vẽ màu đỏ */
	setlinestyle(SOLID_LINE, 0, 3); /* nét vẽ liền và dày 3 pixel */
	setfillstyle(SOLID_FILL, BLUE); /* tô kín hình vẽ bằng màu xanh da trời */
	pieslice(getmaxx()/2, getmaxy()/2, 0, 360, 100);
	getch();
	closegraph(); /* đóng chế độ đồ hoạ */
}
/* ************************************************************************************************ */
```

6.3.4. Vẽ đường thẳng

1. Hàm line

Dạng hàm: void line(int x1, int y1, int x2, int y2);

Công dụng: hàm vẽ ra một đường thẳng nối hai điểm (*x1, y1*) và (*x2, y2*) trên màn hình đồ hoạ (*hàm không di chuyển vị trí con trỏ màn hình*). Hàm này thường được sử dụng với hàm void *moveto (int x, int y)* nhằm di chuyển con trỏ màn hình tới vị trí (*x,y*).

2. Hàm lineto

Dạng hàm: void lineto(int x, int y);

Công dụng: hàm vẽ ra một đường thẳng từ vị trí hiện tại của con trỏ màn hình đến điểm (*x,y*) và di chuyển con trỏ đến điểm (*x, y*).

3. Hàm linerel

Dạng hàm: void linerel(int dx, int dy);

Công dụng: hàm vẽ ra một đường thẳng từ vị trí hiện tại (*x,y*) của con trỏ màn hình đến điểm (*x+dx, y+dy*) và di chuyển con trỏ đến vị trí mới.

6.3.5. Vẽ hình chữ nhật

1. Hàm rectangle

Dạng hàm: void rectangle(int x1, int y1, int x2, int y2);

Công dụng: hàm vẽ ra một đường chữ nhật có các cạnh song song với các cạnh của màn hình, có toạ độ góc trên trái là (*x1, y1*) và toạ độ góc dưới phải là (*x2, y2*).

2. Hàm bar

Dạng hàm: void bar(int x1, int y1, int x2, int y2);

Công dụng: hàm vẽ và tô màu một hình chữ nhật có toạ độ góc trên trái là (*x1,y1*) và toạ độ góc phải dưới là (*x2, y2*). Hàm không ảnh hưởng bởi *setcolor* và *setlinestyle*.

6.3.6. Vẽ đường gấp khúc và đa giác

1. Hàm drawpoly

Muốn vẽ một đường gấp khúc đi qua *n* điểm (*x1, y1*), (*x2, y2*), ...(*xn, yn*) thì trước tiên ta cần gán lần lượt các toạ độ này cho một mảng nguyên *a* nào đó (*a[0]=x1, a[1]=y1, a[2]=x2, a[3]=y2*...). Sau đó thực hiện lời gọi hàm:

drawpoly(n, a);

Khi điểm cuối (*xn, yn*) trùng với điểm đầu (*x1, y1*) ta nhận được một đa giác.

2. Hàm fillpoly

Giả sử ta đã có mảng các số nguyên chứa toạ độ của n điểm như ở trên, nếu ta thực hiện lời gọi hàm: *fillpoly(n, a);* thì ta sẽ nhận được một *hình đa giác* tô màu *n* đỉnh tương ứng với các toạ độ (*x1, y1*), (*x2, y2*), ...(*xn, yn*).

6.3.7. Các hàm xử lí điểm ảnh

1. Hàm putpixel

Dạng hàm: void putpixel(int x, int y, int Mau);

Công dụng: hàm sẽ tô điểm (*x, y*) trên màn hình bằng màu xác định bởi *Mau*.

2. Hàm getpixel

Dạng hàm: unsigned getpixel(int x, int y);

Công dụng: hàm sẽ trả về giá trị màu của điểm ảnh (*x,y*) trên màn hình. Hàm sẽ cho giá trị bằng 0 nếu điểm (*x, y*) chỉ là mầu nền (*chưa được vẽ bởi các hàm khác*).

3. Hàm floodfill

Dạng hàm: void floodfill(int x, int y, int MauBien);

Công dụng: Hàm sẽ tô màu cho một miền kín có màu biên bằng màu của *MauBien* và điểm gieo (*x,y*) nằm trong miền kín này. Nếu điểm gieo nằm ngoài miền kín đó thì vùng ngoài của miền kín được tô màu. Còn nếu không tồn tại một miền kín như vậy thì toàn bộ màn hình được tô màu (*màu tô, kiểu tô xác định bởi setfillstyle*).

6.3.8. Xử lí toạ độ đề các trên màn hình đồ hoạ

Trong thực tế, nhiều khi ta cần vẽ các hình trong hệ toạ độ đề các lên màn hình đồ hoạ nhưng vẫn đảm bảo tính trung thực của hình vẽ. Tuy nhiên ta không thể sử dụng trực tiếp những toạ độ đã cho trong hệ đề các để vẽ lên màn hình đồ hoạ (*do sự không tương thích về kích cỡ và chiều quy ước của hệ toạ độ*). Để thực hiện được điều này ta cần tiến hành theo các bước dưới đây.

1. Tính hệ số co dãn toạ độ

Giả sử cần vẽ một hình trong toạ độ đề các (*hình này nằm trong hình chữ nhật bao có toạ độ góc trái trên là (Xmax, Ymax) và góc dưới phải là (Xmin, Ymin)*) lên màn hình đồ hoạ trong một cửa sổ có toạ độ góc trái trên là (*x1, y1*) và góc phải dưới là (*x2, y2*) thì các hệ số co dãn toạ được tính như sau (*nếu kích thước hình vẽ không cần theo đúng tỉ lệ với hình thật thì có thể bỏ qua bước 1 và 2 như ví dụ 6-3 dưới đây*):

Kx= (x2-x1)/(Xmax-Xmin) ; /* Kx là hệ số co dãn theo trục x */

Ky = (y2-y1)/(Ymax-Ymin) ; /* Ky là hệ số co dãn theo trục y */

2. Tính toạ độ trong hệ đề các sau khi đã co dãn toạ độ

Toạ độ của điểm (*x,y*) bất kì của hình cần vẽ sau khi co dãn (*cho phù hợp kích thước*) được tính theo công thức sau: x' = (int)x*Kx và y'=(int)y*Ky;

Trong đó (*x', y'*) là toạ độ mới của điểm (*x,y*) sau khi đã thực hiện phép co dãn.

3. Đổi toạ độ đề các về toạ độ màn hình đồ hoạ

Toạ độ (*x', y'*) vừa tính được cần được đổi thành toạ độ (X*m*, *Ym*) trong màn hình đồ hoạ trước khi hình được vẽ theo công thức biến đổi sau:

Xm = Xo + x' và Ym = Yo – y'

Trong đó (*Xo, Yo*) là gốc toạ độ của hệ trục đề các đã quy đổi sang toạ độ màn hình và chúng có thể được tính theo công thức:

Xo = x1 + (int) (-Xmin * Kx)
Yo = y1 + (int) (Ymax * Ky)

Ví dụ 6-3. Viết chương trình vẽ một hình ngôi sao *n* cánh (*nét vẽ màu xanh da trời*) và một hình đa giác màu xanh da trời (*nét vẽ màu đỏ*) lên màn hình màu trắng.

```
/* ****************************************************************************************** */
#include "graphics.h"
#include "conio.h"
#include "stdio.h"
#include "math.h"
#include "process.h"
typedef  struct
{
    int x, y;
} DiemAnh;
#define MAX 20 /* số đỉnh tối đa của đa giác */
#define HESO  0.017453293 /* Hệ số chuyển đổi từ độ sang radian */
void NgoiSao(int SoDinh, int BanKinh, DiemAnh Tam, int MauNet);
void DaGiac(int SoDinh, int BanKinh, DiemAnh Tam, int MauNet, int MauTo);
/* ****************************************************************************************** */
int main()
{
        int Driver =DETECT, Mode, MaLoi, SoDinh = 5;
        DiemAnh T1, T2 ;
        initgraph(&Driver, &Mode, "") ;
        MaLoi=graphresult() ;
        if(MaLoi)
        {
            printf(" \nLoi do hoa, ma loi la:  %d\n", MaLoi) ;
            puts(grapherrormsg(MaLoi)) ;
            exit(-1) ;
        }
        setbkcolor(WHITE); /* Đặt màu nền trắng */
        T1.x = getmaxx()/4; /* toạ độ tâm ngôi sao n cánh trên màn hình đồ hoạ */
        T1.y = getmaxy()/2;
        T2.x = 3*getmaxx()/4; /* toạ độ tâm của đa giác  trên màn hình đồ hoạ */
        T2.y = getmaxy()/2 ;
        NgoiSao(SoDinh, 150, T1, BLUE) ;
```

```
        DaGiac(SoDinh, 150, T2, RED, BLUE);
        getch();
        closegraph(); /* đóng chế độ đồ hoạ */
        return 0;
}
/* ******************************************************************************************* */
void NgoiSao(int SoDinh, int BanKinh, DiemAnh Tam, int MauNet)
{
        int i, j;
        DiemAnh  a[MAX];
        double RaDian = M_PI/2;
        for (i=0; i<SoDinh; i++) /* tính toạ độ đề các cho các đỉnh của ngôi sao n cánh */
        {
                a[i].x = BanKinh * cos(RaDian);
                a[i].y = BanKinh * sin(RaDian);
                RaDian += HESO*360/SoDinh;
        }
        setcolor(MauNet);
        for (i=0; i< SoDinh -1; i++)
                for (j= i+1; j<SoDinh; j++)
                        if ( j-i> 1 && ((i!=0) || (j!= SoDinh-1)))
                                line(Tam.x+a[i].x, Tam.y – a[i].y, Tam.x+ a[j].x, Tam.y-
                                a[j].y);
        /* ở đây Tam.x chính là Xo, Tam.y chính là Yo, (a[i].x , a[i].y) là toạ độ của
đỉnh thứ i của hình sao tính trong toạ độ đề các */
}
/* ******************************************************************************************* */
void DaGiac(int SoDinh, int BanKinh, DiemAnh Tam, int MauNet, int MauTo)
{
        int i;
        int a[2*MAX];
        double RaDian = M_PI/2;
        for (i=0; i<2*SoDinh; i+=2) /* tính toạ độ đề các cho các đỉnh của đa giác */
        {
                a[i] = BanKinh * cos(RaDian); /* toạ độ x của đỉnh i trong hệ đề các */
                a[i] = Tam.x + a[i]; /* quy đổi sang toạ độ màn hình */
                a[i+1] = BanKinh * sin(RaDian); /* toạ độ y của đỉnh i */
                a[i+1] = Tam.y - a[i+1]; /* quy đổi sang toạ độ màn hình */
                RaDian += HESO*360/SoDinh;
        }
        setcolor(MauNet);
        setfillstyle(SOLID_FILL, MauTo);
        fillpoly(SoDinh, a); /* vẽ và tô màu cho đa giác */
}
/* ******************************************************************************************* */
```

6.3.9. Xử lí văn bản trên màn hình đồ hoạ

Trong chế độ đồ hoạ, các hàm hiển thị văn bản thông thường như *printf* không thể hoạt động được mà ta phải sử dụng các hàm khác và tự mình điều chỉnh để văn bản có thể được hiển thị trên màn hình đồ hoạ theo như ý muốn.

1. Các hàm hiển thị văn bản trong chế độ đồ hoạ

a) Hàm outtext

Dạng hàm: void outtext(char *P);

Công dụng: Dùng để hiển thị chuỗi kí tự do con trỏ P trỏ tới ra màn hình đồ hoạ tại vị trí hiện thời của con trỏ màn hình.

b) Hàm outtextxy

Dạng hàm: void outtextxy(int x, int y, char *P);

Công dụng: Dùng để hiển thị chuỗi kí tự do con trỏ P trỏ tới ra màn hình đồ hoạ tại toạ độ (*x,y*) của màn hình.

2. Định dạng văn bản ra màn hình đồ hoạ

Để xác định *Font* chữ, cỡ chữ và hướng hiển thị của văn bản ta dùng hàm *settextstyle* theo cú pháp như sau :

settextstyle (int Font, int Huong, int KichThuoc);

Trong đó tham số *Font* có thể nhận một trong các giá trị cho trong bảng 6.5

Bảng 6.5. Các kiểu Font chữ trong chế độ đồ hoạ

Tên hằng Font	Giá trị	Tên hằng Font	Giá trị
DEFAULT_FONT	0	SANS_SERIF_FONT	3
TRIPLEX_FONT	1	GOTHIC_FONT	4
SMALL_FONT	2		

Mỗi kiểu Font ở trên sẽ tương ứng với một tệp tin *.*CHR* của Turbo C. Nếu các tệp tin này không tồn tại thì hàm không có tác dụng.

Tham số *Huong* có thể nhận một trong hai giá trị là *HORIZ_DIR* (*văn bản sẽ được hiển thị theo chiều ngang từ trái qua phải*) hoặc *VERT_DIR* (*văn bản sẽ được hiển thị theo chiều thẳng đứng từ dưới lên*). Còn tham số *KichThuoc* là một số nguyên có giá trị từ *1* đến *10* để định ra hệ số phóng đại của chữ.

Để xác định vị trí tương đối của văn bản so với toạ độ (*x,y*) trong hàm *outtextxy* hoặc so với vị trí hiện thời của con trỏ màn hình trong hàm *outtext* ta dùng hàm: void settextjustify(int ChieuNgang, int ChieuDoc);

Trong đó tham số *ChieuNgang* và *ChieuDoc* có thể nhận các giá trị sau:

Bảng 6.6. Các giá trị xác định vị trí tương đối để hiển thị văn bản

ChieuNgang	Mô tả	ChieuDoc	Mô tả
LEFT_TEXT (0)	Văn bản sẽ xuất hiện bắt đầu từ toạ độ (x,y).	BOTTOM_TEXT(0)	Toạ độ (x,y) sẽ nằm bên dưới văn bản.
CENTER_TEXT(1)	Toạ độ (x,y) sẽ nằm giữa tâm chiều dài của văn bản	CENTER_TEXT(1)	Toạ độ (x,y) sẽ nằm chính giữa chiều cao.
RIGHT_TEXT (2)	Toạ độ (x,y) sẽ nằm ở cuối của văn bản	TOP_TEXT(2)	Toạ độ (x,y) sẽ nằm bên trên của văn bản.

3. Xác định chiều cao và chiều rộng của văn bản

Để có thể xác định được chiều cao của văn bản ta dùng hàm:

*int textheight (char *P);* Hàm sẽ trả về số điểm ảnh theo chiều cao của văn bản do con trỏ *P* trỏ tới.

Để có thể xác định được chiều rộng của văn bản ta dùng hàm:

*int textwidth (char *P);* Hàm sẽ trả về số điểm ảnh theo chiều dài của văn bản do con trỏ *P* trỏ tới.

Ví dụ 6-4. Viết chương trình hiện ra dòng chữ "DAI HOC BACH KHOA HA NOI" màu xanh da trời theo chiều ngang vào chính giữa màn hình màu trắng.

```
/* ***************************************************************************** */
#include "graphics.h"
#include "conio.h"
#include "stdio.h"
#include "process.h"
/* ***************************************************************************** */
int main()
{
      int Driver = DETECT, Mode, MaLoi;
      initgraph(&Driver, &Mode, "") ;
      MaLoi=graphresult() ;
      if(MaLoi)
      {
          printf(" \nLoi do hoa, ma loi la:  %d\n", MaLoi) ;
          puts(grapherrormsg(MaLoi)) ;
          exit(-1) ;
      }
      setbkcolor(WHITE); /* Đặt màu nền trắng */
      setcolor(BLUE); /* Đặt nét chữ màu xanh */
      settextstyle(DEFAULT_FONT, HORZ_DIR, 2);
      settextjustify(CENTER_TEXT, CENTER_TEXT) ;/*(x,y) nằm chính giữa văn bản*/
      outtextxy(getmaxx()/2, getmaxy()/2, "DAI HOC BACH KHOA HA NOI");
      getch();
      closegraph(); /* đóng chế độ đồ hoạ */
      return 0;
}
/* ***************************************************************************** */
```

6.4. KĨ THUẬT TẠO HÌNH CHUYỂN ĐỘNG

Có nhiều phương pháp khác nhau để tạo ra các hình chuyển động trong chế độ đồ hoạ như phương pháp vẽ xoá, phương pháp lật trang, phương pháp lưu giữ ảnh vào bộ nhớ động… Trong giáo trình này chúng tôi xin giới thiệu phương pháp vẽ và xoá các hình lân cận nhau tạo hiệu ứng 24 hình / giây. Để xoá một hình ta vẽ lại hình đó với màu nền.

Ví dụ 6-5. Viết chương trình vẽ một quả bóng chuyển động trong một hình chữ nhật.

```
/* ***************************************************************************** */
#include "graphics.h"
#include "conio.h"
#include "stdio.h"
#include "process.h"
#include "stdlib.h"
#include "dos.h"
/* ***************************************************************************** */
int main()
{
      int Driver = DETECT, Mode, MaLoi;
```

```
    int x1=10, y1=10, x2, y2, x, y, dx, dy, R ;
    initgraph(&Driver, &Mode, "") ;
    MaLoi=graphresult() ;
    if(MaLoi)
    {
        printf(" \nLoi do hoa, ma loi la:  %d\n", MaLoi) ;
        puts(grapherrormsg(MaLoi)) ;
        exit(-1) ;
    }
    x2=getmaxx()-10 ; y2= getmaxy()-10 ; /* Xác định khung chữ nhật */
    setbkcolor(BLUE); /* Đặt màu nền xanh da trời */
    setcolor(WHITE) ; moveto(x1, y1) ; /* chuyển con trỏ đến toạ độ (x1, y1) */
    lineto(x2,y1); lineto(x2,y2); lineto(x1,y2); lineto(x1,y1); /* vẽ khung chữ nhật */
    dx=10; dy=10; R=5; /* Bước chuyển động và bán kính quả bóng */
    x1= x1+R+14; y1=y1+R+14; x2=x2-R-14; y2=y2-R-14;
    x=50+random(getmaxx()-50) ; y=30+random(getmaxy()-50) ;
    do
    {
        setcolor(WHITE); circle(x, y, R); /* Vẽ quả bóng */
        delay(1000); setcolor(getbkcolor()); /* lấy màu nền hiện tại */
        circle(x, y, R); delay(1000); /* Xoá quả bóng */
        if((x>x2)||(x<x1))  dx = -dx ; /* Đổi hướng chuyển động */
        if((y>y2)||(y<y1))  dy = -dy;
        x += dx ; y += dy ; /* Chuẩn bị vẽ hình tiếp theo ở vị trí kế cận */
    } while (!kbhit());
    getch();
    closegraph(); /* đóng chế độ đồ hoạ */
    return 0;
}
/* ************************************************************************************************ */
```

6.5. CÂU HỎI VÀ BÀI TẬP

1. Làm việc trong chế độ đồ hoạ có những đặc trưng gì? Ta có thể làm được gì?
2. Viết chương trình vẽ một bánh xe lăn trên một đường thẳng nằm ngang.
3. Viết chương trình mô phỏng dao động của một con lắc đơn.
4. Viết chương trình vẽ đồ thị của hàm số y= f(x). Vận dụng với y = cos(x).
5. Viết chương trình cho phép nhận và trình bày kí tự trên màn hình đồ hoạ. Nếu gõ sai, có thể dùng phím BackSpace để xoá kí tự gõ nhầm đó.
6. Viết chương trình vẽ hình hoa hồng f(φ) = K cos(n φ), n là số cánh hoa hồng.
7. Viết chương trình cho một dòng chữ chạy ngang màn hình trong chế độ đồ hoạ.
8*. Viết chương trình vẽ một *Menu* hai cấp ra màn hình đồ hoạ.
9*. Viết chương trình chơi cờ ca rô trong chế độ đồ hoạ.
10*. Viết chương trình minh hoạ một quả bóng nảy trên màn hình (có đàn hồi).
11. Viết chương trình mô phỏng đường bay của đạn đại bác.

Chương 7

MỘT SỐ BÀI TẬP TỔNG HỢP

MỤC TIÊU CỦA CHƯƠNG NÀY

- *Củng cố và hoàn thiện các kiến thức lí thuyết đã được học.*
- *Người học có khả năng sử dụng phối kết hợp các kiến thức lí thuyết đã học để giải quyết một số lớp các bài toán thường gặp từ đó dần dần hình thành tư duy lập trình cho người học.*

7.1. BÀI TOÁN MÔ PHỎNG

Ví dụ 7-1. Hãy viết chương trình mô phỏng đàn *Piano* trên máy tính. Người sử dụng có thể đánh bản nhạc bất kì bằng cách bấm các phím :

A tương ứng với nốt La. B tương ứng với nốt Si. C tương ứng với nốt Đô.

D tương ứng với nốt Rê. E tương ứng với nốt Mi. F tương ứng với nốt Fa và cuối cùng là G tương ứng với nốt Sol.

Mỗi khi cần chuyển từ quãng *8* này sang quãng *8* kế tiếp ta bấm thêm phím *Alt* (*ví dụ từ La trung lên La thanh ta bấm Alt - A*), ngược lại muốn chuyển từ quãng *8* này về quãng *8* kế trước ta bấm thêm phím *Shift* (*ví dụ từ La trung về La trầm ta bấm Shift - A*).

Giải: Để phát ra âm thanh trong ngôn ngữ lập trình C ta dùng hàm *sound(TanSo)*; hàm này khai báo trong *dos.h*. Mỗi khi hàm được gọi, loa của PC sẽ phát ra một âm thanh tương ứng có tần số đúng bằng tham số *TanSo* và nó chỉ dừng kêu khi hàm *nosound()* được gọi. Trong âm nhạc, để phát ra được một nốt nhạc nào đó ta phải gọi hàm *sound* với tham số bằng chính tần số của nốt nhạc đó (*ví dụ muốn phát ra một nốt La trung ta gọi hàm sound(880)*) và tần số của các âm trong quãng *8* tiếp theo sẽ gấp hai lần tần số của các âm tương ứng trong quãng 8 trước đó (*ví dụ tần số của nốt La trung là 880 Hz thì tần số của nốt La trầm là 440 Hz và của nốt La thanh là 1760 Hz...*). Do đó trong chương trình ta chỉ cần lưu trữ tần số của các nốt nhạc trong một quãng 8 là đủ. Để chuyển đến quãng 8 tiếp theo ta bấm kết hợp với phím *Alt* ngược lại ta bấm *Shift*. Chương trình mặc định là đang hoạt động với các âm trung. Để thay đổi ta có thể bấm -> (*phím mũi tên phải*) để chuyển sang làm việc với quãng *8* tiếp theo, ngược lại bấm <- (*phím mũi tên trái*).

```
/* ******************************************************************************** */
#include "conio.h"
#include "stdio.h"
#include "dos.h"
enum NoNhac {DO, RE, MI, FA, SOL, LA, SI};
int TANSO[7]= {523, 587, 659, 698, 783, 880, 988} ;
void TaoAm(float HeSo, int TanSo);
```

```
/* ************************************************************************** */
int main()
{
	int ch1, ch2;
	float HeSo=1.;
	clrscr();
	while(1)
	{
		if(!kbhit())
		{
			ch1=getch();
			if(ch1==0) /* tổ hợp phím được bấm */
			{
				ch2=getch();
				switch(ch2)
				{
					case 30:TaoAm(HeSo, TANSO[LA]*2);break; /*Alt-A*/
					case 18: TaoAm(HeSo, TANSO[MI]*2); break; /*Alt-A*/
					case 33: TaoAm(HeSo, TANSO[FA]*2); break; /*Alt-A*/
					case 34: TaoAm(HeSo, TANSO[SOL]*2); break; /*Alt-A*/
					case 48: TaoAm(HeSo, TANSO[SI]*2); break; /*Alt-A*/
					case 46: TaoAm(HeSo, TANSO[DO]*2); break; /*Alt-A*/
					case 32: TaoAm(HeSo, TANSO[RE]*2); break; /*Alt-A*/
					case 77: if (HeSo <= 4) /* Nếu phím <- được bấm*/
						HeSo *=2; break; /* chuyển sang quãng 8 tiếp */
					case 75: if (HeSo >= 1/4) /* Nếu phím -> được bấm */
						HeSo /=2; break;/* chuyển về quãng 8 trước đó*/
					default: nosound();
				}
			}
			else
			{
				switch(ch1)
				{
					case 27: nosound(); return 0; /* Bấm ESC thoát*/
					case 'a': TaoAm(HeSo, TANSO[LA]); break; /* Bấm a*/
					case 'e': TaoAm(HeSo, TANSO[MI]); break; /* Bấm e*/
					case 'f': TaoAm(HeSo, TANSO[FA]); break; /* Bấm f*/
					case 'g': TaoAm(HeSo, TANSO[SOL]); break; /* Bấm g*/
					case 'b': TaoAm(HeSo, TANSO[SI]); break; /* Bấm b*/
					case 'c': TaoAm(HeSo, TANSO[DO]); break; /* Bấm c*/
					case 'd': TaoAm(HeSo, TANSO[RE]); break; /* Bấm d*/
					case 'A': TaoAm(HeSo, TANSO[LA]/2); break; /* Shift-a*/
					case 'E': TaoAm(HeSo, TANSO[MI]/2); break; /* Shift-e*/
					case 'F': TaoAm(HeSo, TANSO[FA]/2); break; /* Shift-f*/
					case 'G': TaoAm(HeSo,TANSO[SOL]/2); break;/*Shift-g*/
					case 'B': TaoAm(HeSo, TANSO[SI]/2); break; /* Shift-b*/
					case 'C': TaoAm(HeSo, TANSO[DO]/2); break; /* Shift-c*/
					case 'D': TaoAm(HeSo, TANSO[RE]/2); break; /* Shift-d*/
					default: nosound();
				}
```

```
                }
            }
        }
        return 0;
}
/* ************************************************************************** */
void TaoAm(float HeSo, int TanSo)
{
        int TG;
        TG=(int) (HeSo*TanSo);
        nosound(); /* Tắt âm trước đó */
        sound(TG); /* Phát ra âm thanh theo tần số chỉ định */
}
/* ************************************************************************** */
```

7.2. BÀI TOÁN THỐNG KÊ

Ví dụ 7-2. Hãy viết chương trình xét xem một chương trình nguồn viết bằng một thứ ngôn ngữ nào đó đã sử dụng bao nhiêu từ khoá của ngôn ngữ này và số lần lặp của các từ khoá đó trong chương trình.

Giải: Để giải quyết được bài toán trên ta cần lưu trữ bộ từ khoá của ngôn ngữ trong một tệp (*gọi là tệp từ khoá Keyword.txt*) theo quy cách mỗi dòng một từ và tất cả các chữ cái đều viết bằng chữ thường (*giả thiết có không quá 100 từ khoá và mỗi từ khoá không vượt quá 32 kí tự*). Giữa các từ sẽ được phân cách nhau bởi một số kí tự nhất định (*cho trong mảng KiTuPhanCach*). Chương trình sẽ thực hiện đọc tệp các từ khoá vào mảng *KeyWord* để xử lí, sau đó đọc lần lượt từng *từ văn bản* trong tệp nguồn, tách bỏ những kí tự phân cách ra khỏi từ đó (*do hàm GetWord thực hiện*) rồi đem so sánh nó với các từ khoá trong mảng *KeyWord*. Nếu là từ khoá thì tiến hành tìm kiếm trong một mảng *List* chứa các từ đã được tìm thấy, nếu từ mới nhận có mặt trong *List* thì tiến hành tăng số lượng của nó lên 1, ngược lại tiến hành bổ sung từ khoá mới này vào danh sách từ khoá đã tìm thấy *List* với số lượng là 1. Cuối cùng kết quả nhận được sẽ được ghi ra tệp có tên *OutPutName* theo khuôn dạng: *Tên từ khoá tìm thấy   số lần xuất hiện*.

```
/* ************************************************************************** */
#include "conio.h"
#include "stdio.h"
#include "dos.h"
#include "process.h"
#include "string.h"
#define MAX 100 /* Số từ khoá tối đa */
#define MAXTEN 32
char KiTuPhanCach[]={' ', '.', ':', ',', ';', '=', '(', ')', '[', ']', '{', '}', '\'', '"', '\0'};
typedef struct
{
        char Ten[MAXTEN]; /* Tên từ khoá */
        int n; /* Số lần xuất hiện */
} RecType;
int nKeyWord, nList;
char KeyWord[MAX][MAXTEN];
```

```
RecType List[MAX];
char InPutName[MAXTEN], OutPutName[MAXTEN];
void ReadKeyWord(char *KeyWordName);
char * GetWord(char * s, int * i);
int IsKeyWord(char *w);
int InList(char *w);
void CreatList(char *name);
void WriteResult(char *name);
/* ************************************************************************* */
int main()
{
      printf("\n Hay nhap ten tep nguon");
      gets(InPutName);
      printf("\n Hay nhap ten tep dich");
      gets(OutPutName);
      ReadKeyWord("KeyWord.txt");
      CreatList(InPutName);
      WriteResult(OutPutName);
      return 0;
}
/* ************************************************************************* */
/* Đọc các từ khoá trong tệp có tên KeyWordName vào mảng từ khoá KeyWord */
void ReadKeyWord(char *KeyWordName)
{
      FILE *F;
      F=fopen(KeyWordName, "r");
      if(F==NULL) /*Nếu không mở được tệp*/
      {
            printf("\nKhong mo duoc tep %s", KeyWordName);
            getch(); /*Dừng chương trình để xem thông báo */
            exit(-1); /*Thoát*/
      }
      nKeyWord=-1;
      while(!feof(F))
      {
            nKeyWord++;
            fscanf(F, "%s", KeyWord[nKeyWord]);
            if(ferror(F)) /*Nếu có lỗi trong lúc đọc */
            {
                  perror("Loi doc tep : ");
                  fclose(F);
                  exit(-1);
            }
      }
      fclose(F);
}
/* ************************************************************************* */
/* Trả về một từ được tách ra từ xâu s tại vị trí i, sau đó trả về vị trí kế tiếp*/
char * GetWord(char * s, int * i)
{
      int L, j=0;
```

```
    char w[MAXTEN];
    L= strlen(s);
    /* Bỏ qua kí tự phân cách */
    while((*i<L)&&(strchr(KiTuPhanCach, s[*i])!=NULL)) ++(*i);
    /* Tích luỹ dần cho w */
    while((*i<L)&&( strchr(KiTuPhanCach, s[*i])==NULL))
    {
        w[j] = s[*i]; ++j; ++(*i);
    }
    w[j] = '\0'; /* Đánh dấu sự kết thúc xâu w */
    return (w);
}
/* ********************************************************************************* */
/*Kiểm tra w có phải là từ khoá hay không, đúng cho giá trị 1, ngược lại cho giá trị 0*/
int IsKeyWord(char *w)
{
    int i=0;
    /* Tìm từ khoá đầu tiên trùng với w */
    while((i<nKeyWord)&&(strcmp(KeyWord[i], w)!=0))  ++i;
    return (i<nKeyWord);
}
/* ********************************************************************************* */
/* Tìm kiếm w trong List */
int InList(char *w)
{
    int i=0;
    while((i<=nList)&&(strcmp(List[i].Ten, w)!=0))  ++i;
    return (i);
}
/* ********************************************************************************* */
/* Tạo mảng List chứa các từ khoá của tệp có tên name */
void CreatList(char *name)
{
    FILE *F;
    char *w, s[MAXTEN];
    int i, j;
    F=fopen(name, "r");
    if(F==NULL) /*Nếu không mở được tệp*/
    {
        printf("\nKhong mo duoc tep %s", name);
        getch(); /*Dừng chương trình để xem thông báo */
        exit(-1); /*Thoát*/
    }
    nList=-1;
    while(!feof(F))
    {
        fscanf(F, "%s", s);
        if(ferror(F)) /*Nếu có lỗi trong lúc đọc */
        {
            perror("Loi doc tep : ");
            fclose(F);
```

```
			exit(-1);
		}
		i=0;
		w= GetWord(s, &i); /* tách trong xâu s từ vị trí i một từ w */
		while(strcmp(w, "")!=0)
		{
			if(IsKeyWord(w)) /* Nếu là từ khoá */
			{
				j = InList(w);
				if(j>nList) /* nếu w không thuộc danh sách đã có */
				{
					++nList; /* Bổ sung w vào cuối danh sách từ khoá */
					strcpy(List[nList].Ten, w); /* Đưa w vào danh sách */
					List[nList].n=1; /*đếm số lượng từ khoá trong danh sách*/
				}
				else /* Từ khoá đã có trong danh sách */
				{
					++List[j].n; /* Đếm số lần xuất hiện của từ khoá */
				}
			}
			w=GetWord(s, &i); /* Tách từ tiếp theo */
		}
	}
	fclose(F);
}
/* ************************************************************************** */
/* Ghi List lên tệp văn bản có tên name theo quy cách */
void WriteResult(char *name)
{
	FILE *F;
	int i;
	F=fopen(name, "wt");
	if(F==NULL) /*Nếu không mở được tệp*/
	{
		printf("\nKhong mo duoc tep %s", name);
		getch(); /*Dừng chương trình để xem thông báo */
		exit(-1); /*Thoát*/
	}
	for(i=0; i<=nList; ++i)
	{
		fprintf(F, "%-16s%3d\n", List[i].Ten, List[i].n);
		if(ferror(F)) /*Nếu có lỗi trong lúc đọc */
		{
			perror("Loi doc tep : ");
			fclose(F);
			exit(-1);
		}
	}
	fclose(F);
}
/* ************************************************************************** */
```

7.3. BÀI TOÁN VẼ ĐỒ THỊ HÀM SỐ

Ví dụ 7-3. Viết chương trình vẽ toạ độ của hàm $y=f(x)$ trên đoạn [a, b] lên một cửa sổ bất kì của màn hình. Sử dụng chương trình đó vẽ đồ thị cho hàm $y=3x^5+2x^3-6x^2+1$ trên đoạn [-100, 100].

Giải: Ta cần thực hiện các bước sau đây:

- *Tìm hình chữ nhật chứa hình cần vẽ :* ***Xmax-Xmin= b-a và Ymax-Ymin.***
- *Co dãn hệ số toạ độ để đồ thị nằm trọn trong cửa sổ cần vẽ .*
- *Đổi từ toạ độ đề các sang toạ độ của màn hình.*

```
/* ********************************************************************************* */
#include "graphics.h"
#include "process.h"
#include "conio.h"
#include "stdio.h"
#include "math.h"
void Menu(void);
void VeDoThi(int x1, int y1, int x2, int y2,double (*f)(double), double a, double b);
void ThongBao(int x, int y) ;
double F(double x);
void MinMax(double (*f)(double), double a, double b, double * Max, double *Min);
/* ********************************************************************************* */
int  main()
{
        int Driver = DETECT, Mode, MaLoi, x1, x2, y1, y2;
        double a, b ;
        Menu() ;
        printf("\n Hay nhap khoang a, b= " );
        scanf("%lf%lf", &a, &b);
        printf("\nHay nhap toa do man hinh cua ca so la x1, y1, x2, y2 = ");
        scanf("%d%d%d%d", &x1, &y1, &x2, &y2);
        initgraph(&Driver, &Mode, "") ;
        MaLoi=graphresult() ;
        if(MaLoi)
        {
             printf(" \nLoi do hoa, ma loi la:  %d\n", MaLoi) ;
             puts(grapherrormsg(MaLoi)) ;
             exit(-1) ;
        }
        setbkcolor(BLUE);
        VeDoThi(x1, y1, x2, y2, F, a, b);
        ThongBao(x1, y1);
        getch();
        closegraph();
        return 0;
}
/* ********************************************************************************* */
void Menu(void)
{
```

```
    clrscr() ;
    printf("\n ******* CHUONG TRINH VE DO THI HAM SO Y=F(x) ************ ") ;
    printf("\n\n Ap dung cho ham y=3x^5+2x^3-6x^2+1 tren doan [-100, 100]");
    printf("\n\nAn phim bat ki de chay tiep");
    getch();
}
/* ************************************************************************** */
void VeDoThi(int x1, int y1, int x2, int y2, double (*f)(double), double a, double b)
{
    double x, y, kx, ky, Ymax, Ymin, dx ;
    int i, X0, Y0, Xm, Ym ;
    setfillstyle(1, LIGHTRED);
    bar(x1, y1, x2, y2); /* vẽ cửa sổ */
    setcolor(3);
    moveto(x1, y1); lineto(x2, y1); lineto(x2, y2); lineto(x1, y2); lineto(x1, y1);
    MinMax(f, a, b, &Ymin, &Ymax);
    kx=(x2-x1) / (b-a); ky= (y2-y1)/(Ymax-Ymin) ;
    setcolor(RED);
    X0= x1+(int) (-a*kx);  Y0= y1+(int) (Ymax*ky);
    /* Vẽ hai trục*/
    setcolor(RED); line(x1, Y0, x2, Y0);  line(X0, y1, X0, y2);
    x=a; Xm= X0+ (int) (x*kx); y=f(x); Ym= Y0-(int)(y*ky);
    moveto(Xm, Ym); dx= (b-a)/(x2-x1); setcolor(WHITE);
    while(x<b)
    {
        x += dx; y=f(x);
        Xm= X0+(int)(x*kx); Ym= Y0 - (int)(y*ky);
        lineto(Xm, Ym);
    }

}
/* ************************************************************************** */
void MinMax(double (*f)(double), double a, double b, double * Min, double *Max)
{
    double x, y, dx ;
    x=a ; y=f(x) ; *Max=y ; *Min=y ; dx=(b-a)/640 ;
    while(x<b)
    {
        x += dx ; y=f(x);
        if(y<*Min)  *Min=y;
        if(y>*Max)  *Max =y ;
    }
}
/* ************************************************************************** */
void ThongBao(int x, int y)
{
    setcolor(RED);
    settextstyle(DEFAULT_FONT, HORIZ_DIR, 1);
    outtextxy(x, y, "Bam phim bat ki de tiep tuc ");
    getch() ;
}
```

```
/* ******************************************************************************** */
double F(double x) /* ở đây có thể gán cho bất kì hàm nào */
{
    double Tam = 3*pow(x, 5)+ 2*pow(x, 3) - 6*x*x +1;
    return (Tam);
}
/* ******************************************************************************** */
```

7.4. BÀI TOÁN ĐỆ QUY

Ví dụ 7-4. Viết chương trình mô phỏng bài toán tháp Hà Nội trong chế độ đồ hoạ.

Mô tả bài toán: *có n đĩa kích thước nhỏ dần (có lỗ ở giữa) có thể xếp chồng lên nhau xuyên qua một cọc sao cho đĩa to được đặt ở dưới, đĩa nhỏ được đặt ở trên. Ban đầu chồng đĩa được đặt ở cột A. Hãy mô phỏng các bước chuyển đĩa từ cột A sang cột C theo nguyên tắc:*

- *Mỗi lần chỉ được chuyển đúng một đĩa.*
- *Không bao giờ đĩa to được đặt lên trên đĩa nhỏ hơn.*
- *Được phép sử dụng một cọc trung gian.*

Đây là một bài toán rất khó nếu chúng ta không sử dụng giải thuật đệ quy. Tuy nhiên vấn đề sẽ trở nên đơn giản hơn nếu ta để ý rằng:

- Nếu chỉ có một đĩa thì ta chỉ việc chuyển đĩa từ cột A sang cột C.

- Nếu có hai đĩa ta sẽ thực hiện các thao tác chuyển đĩa như sau:

+ *Chuyển đĩa trên cùng từ cột A sang cột trung gian.*

+ *Chuyển đĩa dưới cùng từ cột A sang cột C.*

+ *Chuyển đĩa nhỏ từ cột trung gian sang cột C.*

- Nếu số đĩa từ 3 trở lên thì vấn đề bắt đầu trở nên phức tạp hơn nhiều. Tuy nhiên ta có thể đưa trường hợp này về hai trường hợp đầu bằng cách coi n-1 đĩa ở trên cùng như là một đĩa và ta có thể quy về trường hợp hai đĩa như sau:

+ *Chuyển n-1 đĩa trên cùng sang cột trung gian.*

+ *Chuyển đĩa thứ n sang cột C.*

+ *Chuyển n-1 đĩa từ cột trung gian về cột C.*

Và như vậy bài toán có thể được viết đệ quy như dưới đây (*dữ liệu được tổ chức dưới dạng danh sách móc nối*).

```
/* ******************************************************************************** */
#include "conio.h"
#include "stdio.h"
#include "alloc.h"
#include "graphics.h"
#include "process.h"
#include "dos.h"
#define DayMax 10
#define tMax 6000
#define BuocCoc 10
#define CaoHon 3
#define RMax 90
#define RMin 20
#define Mau0 0
#define MauNen 1
```

```
#define DichPhai 50
#define DichTrai 50
/* ************************************************************************************ */
typedef struct dd
{/* Mỗi đĩa được đặc trưng bởi số hiệu đĩa (để biết đĩa to hay nhỏ), vị trí của đĩa (để
biết vị trí hiện tại của mỗi đĩa) và màu sắc của đĩa (các đĩa sẽ có màu khác nhau )*/
	int shieudia;
	int vitridia;
	int mau;
	struct dd *next;
} DiaPoint;
typedef struct
{/* Mỗi cột được đặc trưng bởi các tham số là số hiệu cột (để biết là cột A, B hay C),
số lượng các đĩa hiện có trên cột và con trỏ trỏ đến đĩa dưới cùng (chính là con trỏ
đầu của danh sách)*/
	int shieucoc;
	int c;
	DiaPoint *next;
} Coc;

int  T1,T2,T3, Y0, Yd, X,Y;
Coc    A, B, C; /* Khai báo ra 3 cột */
DiaPoint   *Tam;
int   n, gd, gm, buoc, day;
void tao_dia(Coc A, DiaPoint *P);
void tao_dia_t(Coc A, DiaPoint *P);
void xoa_dia(Coc A, DiaPoint *P);
void khoi_tao(Coc *A, Coc *B, Coc *C);
void chuyen_dia(Coc * A, Coc *B, DiaPoint *P);
void thap_hn(Coc *A, Coc *B, Coc *C, DiaPoint*P);
int main()
{
	A.next=NULL;
	B.next=NULL;
	C.next=NULL;
	khoi_tao(&A, &B, &C);
	Tam=A.next;
	getch();
	thap_hn(&A, &B, &C,Tam);
	getch();
	closegraph();
	return 0;
}

/* ************************************************************************************ */
void tao_dia(Coc A, DiaPoint *P)
{
	int x1,x2,y1,y2,mau;
	mau=P->mau;
	setfillstyle(SOLID_FILL,mau);
	x1=((A.shieucoc-RMax)+(P->shieudia-1)*buoc);
	y1=(Yd-P->vitridia*day);
```

```
        x2=((A.shieucoc+RMax)-(P->shieudia-1)*buoc);
        y2=(Yd-(P->vitridia-1)*day-1);
        bar(x1,y1,x2,y2);
}
/* ******************************************************************************** */
void tao_dia_t(Coc A, DiaPoint *P)
{
        tao_dia(A, P);
        delay(tMax);
}
/* ******************************************************************************** */
void xoa_dia(Coc A, DiaPoint *P)
{
        int x1, x2, y1, y2;
        int   mau;
        mau=MauNen;
        setfillstyle(SOLID_FILL,mau);
        x1=((A.shieucoc-RMax)+(P->shieudia-1)*buoc);
        y1=(Yd-P->vitridia*day);
        x2=((A.shieucoc+RMax)-(P->shieudia-1)*buoc);
        y2=(Yd-(P->vitridia-1)*day-1);
        bar(x1,y1,x2,y2);
}
/* ******************************************************************************** */
void khoi_tao(Coc *A, Coc *B, Coc *C)
{
      int i;
      DiaPoint *P, *tam;
      clrscr();
      printf("\n\n\n");
      printf("\nHay nhap so luong dia:  ");
      scanf("%d", &n);
      A->c=n;
      B->c=0;
      C->c=0;
      for( i=1; i<=n ;++i)
      {
          tam=(DiaPoint*)malloc(sizeof(DiaPoint));
          if( A->next==NULL)
          {
                A->next=tam;
                tam->next=NULL;
                P=A->next;
                P->shieudia=i;
                P->vitridia=i;
                P->mau=(int)(i % 14)+2;
          }
          else
          {
                P->next=tam;
                tam->next=NULL;
                P=tam;
```

```
                P->shieudia=i;
                P->vitridia=i;
                P->mau=(int)(i % 14)+2;
          }
     }
     gd=DETECT;
     initgraph(&gd, &gm, "");
     if(graphresult()!=grOk)
     {
          printf("Loi do hoa");
          exit(-1);
     }
     setbkcolor(MauNen);
     X=getmaxx();
     Y=getmaxy();
     T1=(int)(X/4); /* T1, T2 và T3 chứa toạ độ x của các cột */
     T2=2*T1;
     T3=3*T1+ DichPhai;
     T1=T1-DichTrai;
     A->shieucoc=T1;
     B->shieucoc=T2;
     C->shieucoc=T3;
     Yd=(int)(Y/6);
     Y0=2*Yd;
     Yd=5*Yd;
     Y=(int)((Y-Yd)/2);
     if (n>1) buoc=(int)((RMax-RMin)/(n-1));
     else buoc=0;
     day=(int)((Yd-Y0)/n);
     if (day>DayMax) day=DayMax;
     P=A->next;
     for (i=1;i<=n;++i)
     {
          tao_dia(*A,P);
          P=P->next;
     }
     setcolor(RED);
     outtextxy(T1+(int)(T1/4)+DichPhai,Y,"CHUONG TRINH THAP HA NOI");
}
/* ******************************************************************************** */
void chuyen_dia(Coc * A, Coc *B, DiaPoint *P)
{
     int mau,i,j;
     Coc tam;
     int  v;
     DiaPoint *Q,*P1;
     mau=P->mau;
     if( A->next!=P)
     {
          if( A->next==NULL)
          {
               P->next=NULL;
```

```
		}
		else
		{
			P1=A->next;
			while (P1->next!=P)
				P1=P1->next;
			P1->next=NULL;
		}
	}
	else  if (A->next==P)
	{
		A->next=NULL;
		A->c=A->c-1;
	}
	xoa_dia(*A, P);
	tam=*A;
	i=1;
	j=1;
	v=1;
	while (v)
	{
		i=i+1;
		if (A->shieucoc<B->shieucoc)
		{
			if ((tam.shieucoc)>(int)((A->shieucoc+B->shieucoc)/2)) j=j-1;
			else j=j+1;
		}
		else
		{
			if ((tam.shieucoc)<(int)((A->shieucoc+B->shieucoc)/2)) j=j-1;
			else j=j+1;
		}
		P->vitridia=P->vitridia+j*CaoHon;
		if (A->shieucoc>B->shieucoc)
		{
			tam.shieucoc=tam.shieucoc-i*BuocCoc;
			if ((tam.shieucoc-i*BuocCoc)<B->shieucoc)
				v=0;
		}
		else
		{
			tam.shieucoc=tam.shieucoc+i*BuocCoc;
			if ((tam.shieucoc+i*BuocCoc)>B->shieucoc)
				v=0;
		}
		tao_dia_t(tam,P);
		xoa_dia(tam,P);
		P->vitridia=P->vitridia-j*CaoHon;
	}
	Q=B->next;
	if (Q==NULL)
	{
```

```
            P->vitridia=1;
            B->next=P;
            P->next=NULL;
            B->c=B->c+1;
        }
        else  if (Q!=NULL )
        {
            while( Q->next!=NULL)
            Q=Q->next;
            P->vitridia=Q->vitridia+1;
            P->next=NULL;
            B->c=B->c+1;
            Q->next=P;
        }
        tao_dia_t(*B,P);
}
/* ********************************************************************************** */
void thap_hn(Coc *A, Coc *B, Coc *C, DiaPoint *P)
{
        DiaPoint *Q,*Q1;
        if (A->next!=NULL)
        {
                if (P->next==NULL)
                {
                        Q=A->next;
                        while (Q!=P)
                              Q=Q->next;
                        Q->next=NULL;
                        A->c=A->c-1;
                        chuyen_dia(A, C, P);
                }
                else
                {
                        Q=P;
                        P=P->next;
                        Q1=B->next;
                        if (Q1!=NULL)
                        {
                              while( Q1->next!=NULL)
                              Q1=Q1->next;
                        }
                        thap_hn(A, C, B, P);
                        chuyen_dia(A, C, Q);
                        if (Q1!=NULL)
                              Q1=Q1->next;
                        else
                              Q1=B->next;
                         thap_hn(B, A, C, Q1);
                }
        }
}
/* ********************************************************************************** */
```

PHỤ LỤC

PHỤ LỤC I. NHỮNG KĨ THUẬT VIẾT CHƯƠNG TRÌNH LỚN

Ngôn ngữ lập trình C có nhiều phương tiện khác nhau để xây dựng những chương trình cỡ lớn và phức tạp. Phần này sẽ nêu lên những kĩ thuật cần thiết khi viết các chương trình lớn.

Một chương trình có rất nhiều hàm, có cấu trúc phức tạp không thể viết chung trên một tệp tin để dịch và liên kết ngay được. Việc tách một chương trình lớn và phức tạp thành nhiều tệp tin và có thể do nhiều người khác nhau lập trình sẽ nảy sinh vấn đề dịch từng tệp tin và liên kết như thế nào cho hợp lí nhất. Phương pháp dịch tách biệt và liên kết theo đề án của Turbo C là giải pháp cho vấn đề đó.

Ta hãy xét ví dụ sau:

```
/*Chương trình chính minh họa cách dịch tách biệt*/
#include "stdio.h"
int main()
{
        int  a, b, TongBinhPhuong;
        printf("Cho 2 so nguyen :");
        scanf("%d", &a, &b);
        TongBinhPhuong = tbp(a, b);
        printf("Tong binh phuong la : %d", TongBinhPhuong);
        return 0;
}
```

Chương trình này được viết và ghi vào tệp ***Main.c***.

```
/*Hàm tính tổng bình phương*/
int tbp(int x, int y)
{
    return(x*x+y*y) ;
}
```

Giả sử chương trình này được viết và ghi vào tệp ***tbp.c***.

Vấn đề ở đây là làm thế nào để dịch và liên kết ***tbp.c*** với ***Main.c*** để có chương trình khả thi mong muốn?

1. Chức năng "Project/Make" của Turbo C

Chương trình biên dịch C có hai cách tổ hợp nhiều tệp tin nguồn để liên kết thành tệp tin khả thi. Một là làm "*bằng tay*", tức là đưa các thông báo dịch trên dòng lệnh để dịch từng tệp tin rồi liên kết chúng lại với nhau. Hai là dùng trình tiện ích ***make***.

Cách thứ nhất sẽ tương đối phức tạp khi soạn thảo chương trình từ nhiều tệp tin. Hơn nữa, khi thay đổi nội dung một tệp tin nguồn nào đó, chỉ cần dịch lại tệp

tin này và liên kết lại còn các tệp tin không bị thay đổi thì không cần dịch lại. Khi số lượng tệp tin lớn sẽ rất khó theo dõi công việc này.

Trong Turbo C, trình tiện ích ***make*** đã liên kết với môi trường kết hợp dưới tên gọi "Project/Make". ***Make*** sẽ biết công việc hợp lí phải làm khi ta thông báo cho nó một *danh sách những tệp tin nguồn* cần dịch và liên kết.

2. Tạo tệp tin PRJ

Để dùng được tiện ích ***make***, trước hết phải tạo tệp tin *Project*. Tệp tin này có tên là tên của tệp tin **.exe* cần tạo và phần mở rộng là ***prj***. Chính các tệp tin **.prj* sẽ định hướng cho tiến trình ***make***.

Khi cần kết hợp ***tbp.c*** với ***Main.c*** để có thể nhận được ***TongBP.exe*** ta phải soạn thảo tệp tin ***TongBP.prj*** bằng một trình soạn thảo văn bản nào đó (*dùng luôn trình Edit trong TC*) gồm hai dòng sau đây rồi ghi vào thư mục chủ của Turbo C:

```
Main.c
Tbp.c
```

Sau khi đã có tệp tin **.prj*, ta cần đưa tên của tệp này vào trong *Project name* trong menu *Project* theo các bước sau :

- *Chọn Menu Project bằng cách bấm Alt – P*
- *Chọn Project name (Enter hai lần)*
- *Chọn tên tệp TongBP.prj trong cửa sổ hiện ra (Enter).*

3. Dịch và liên kết theo Make

Bấm F9, môi trường kết hợp sẽ tìm các tệp tin ***Main.c***, ***Tbp.c*** và kiểm tra ngày giờ của chúng. Nếu chưa có các tệp *Main.obj, Tbp.obj* và *TongBp.exe* tương ứng, trình dịch sẽ tạo ra chúng bằng cách dịch các tệp *Main.c* và *Tbp.c* một cách độc lập tạo thành các tệp đối tượng *Main.obj* và *Tbp.obj*, sau đó các tệp này mới được liên kết với nhau một cách hợp lí để tạo ra chương trình khả thi cuối cùng *TongBP.exe* (*đoạn mã tương ứng của hàm tbp() ở tệp tbp.obj sẽ được kết hợp vào TongBP.exe khi hàm này được tham chiếu đến*). Ngược lại các tệp này sẽ được tìm kiếm và kiểm tra ngày giờ so với các tệp tin nguồn, nếu ngày giờ của tệp tin nguồn nào mới hơn thì tệp tin này sẽ được dịch lại. Nếu tệp tin *TongBp.exe* có ngày giờ cũ hơn các tệp tin nguồn thì tiến trình dịch và liên kết được thực hiện lại để tạo ra tệp *EXE* mới hơn.

Khi tiến trình dịch và liên kết kết thúc nếu nhấn *Alt-R* để chạy chương trình thì ta nhận được:

```
Cho 2 so nguyen:   3   4
Tong binh phuong la:  25
```

4. Những ưu điểm của dịch tách biệt

Với một chương trình cỡ lớn có nhiều hàm và hàng ngàn dòng lệnh, việc dịch tách biệt có các ưu điểm sau:

Tại một thời điểm. Lập trình viên chỉ soạn thảo tại một bộ phận nào đó của chương trình. Những phần đã khởi tạo và dịch xong không cần dịch lại, ngoại trừ phần đang soạn thảo, như thế sẽ giảm thiểu thời gian dịch. Ngoài ra, nhiều người có thể cùng tham gia viết một chương trình. Phương pháp dịch tách biệt cho phép kế

hoạch hóa công việc lập trình theo những phần khác nhau của chương trình, tạo cho lập trình viên xây dựng phong cách lập trình tương đối độc lập nhau và làm giảm đáng kể thời gian viết chương trình.

Một điểm nữa là phương pháp dịch tách biệt mở ra triển vọng thiết kế một chương trình theo nhiều môđun có cấu trúc rõ ràng và hợp lí. Song ở đây phát sinh ***vấn đề là làm sao truyền dữ liệu giữa các phần tách biệt*** của một chương trình?

5. Biến ngoài và các tệp tin được dịch tách biệt

Trong ví dụ trên ***Main.c*** và ***tbp.c*** sẽ được liên kết lại, hai số nguyên a, b chuyển từ hàm ***main()*** trong ***Main.c*** cho hàm ***tbp()*** sẽ thông qua đối số của hàm, còn ***tbp()*** chuyển lại cho hàm ***main*** tổng bình phương qua giá trị trả về. Đây là kiểu truyền dữ liệu đơn giản nhất giữa các hàm trong các tệp tin được dịch tách biệt.

Tuy nhiên, một hàm trong một tệp tin có thể truy cập một biến ngoài đặt trong tệp tin khác nếu trong hàm đó có khai báo biến ngoài cần dùng bằng từ khóa ***extern***. Để minh họa ta có thể viết lại chương trình trên như sau:

```
#include "stdio.h"
int a, b; /*Các biến ngoài dùng để trao đổi với hàm tbp() */
int main()
{
        int TongBinhPhuong;
        printf("Cho 2 so nguyen :");
        scanf("%d", &a, &b);
        TongBinhPhuong = tbp(); /* không dùng đối để truyền tham số */
        printf("Tong binh phuong la : %d", TongBinhPhuong);
        return 0;
}
```

Chương trình này không dùng đối số để truyền tham số cho hàm mà sử dụng các biến ngoài ***a, b*** thông qua từ khoá *extern*.

```
int Tbp()
{
        extern a, b; /* báo rằng a, b là hai biến được khai báo ở tệp khác*/
        return (a*a+b*b);
}
```

Trong hàm này không còn sử dụng các đối số làm tham số hình thức mà thay vào đó là khai báo ***extern*** cho hai biến a, b. Từ khóa này báo cho trình biên dịch biết rằng các biến đó đã được định nghĩa ở một tệp tin nào đó khác với tệp tin hiện tại. Do đó, khi thực hiện liên kết những biến loại này sẽ được tìm ra và nối kết hợp lí theo yêu cầu sử dụng. Trong ví dụ trên, các biến ***a, b*** trong ***tbp()*** sẽ được tham chiếu đúng với biến ***a, b*** trong ***Main.c***.

PHỤ LỤC II. CÁC CHỈ THỊ TIỀN XỬ LÍ TRONG NGÔN NGỮ LẬP TRÌNH C VÀ PHƯƠNG PHÁP TỔ CHỨC CÁC TỆP THƯ VIỆN CHƯƠNG TRÌNH

Các chỉ thị tiền xử lí là các lệnh giúp cho việc soạn thảo chương trình nguồn C trước khi biên dịch. Khi dịch một chương trình, không phải chính bản thân chương trình đó được dịch mà trình biên dịch sẽ căn cứ vào các chỉ thị tiền xử lí để chỉnh lại bản gốc. Sau đó, bản chỉnh này mới được biên dịch để chạy.

Trong các phần trước chúng ta đã làm quen với chỉ thị ***#include*** và ***#define***, trong phần này chúng tôi sẽ giới thiệu thêm các chỉ thị biên dịch có điều kiện.

1. Chỉ thị #if dạng 1

```
#if biểu thức hằng
    Đoạn chương trình
#endif
```

Nếu *biểu thức hằng* đúng thì trình biên dịch C sẽ biên dịch đoạn chương trình nằm giữa *#if và #endif*, ngược lại đoạn này sẽ bị bỏ qua.

2. Chỉ thị #if dạng 2

```
#if biểu thức hằng
    Đoạn chương trình 1
#else
    Đoạn chương trình 2
#endif
```

Nếu *biểu thức hằng* đúng thì trình biên dịch C sẽ biên dịch đoạn *chương trình 1* nằm giữa *#if và #else*, ngược lại đoạn *chương trình* 2 sẽ được biên dịch.

3. Chỉ thị dạng #ifdef và #ifndef

Các chỉ thị này có thể có một số cách sử dụng như sau:

Dạng 1:

```
#ifdef tên macro
    Đoạn chương trình
#endif
```

Nếu *tên macro* đã được định nghĩa (*bởi #define*) thì trình biên dịch C sẽ biên dịch đoạn chương trình nằm giữa *#ifdef và #endif*. Ngược lại, đoạn đó bị bỏ qua.

Dạng 2:

```
#ifdef tên macro
    Đoạn chương trình 1
#else
```

Đoạn chương trình 2

#endif

Nếu ***tên macro*** đã được định nghĩa (*bởi #define*) thì trình biên dịch C sẽ biên dịch *đoạn chương trình 1* nằm giữa *#ifdef và #endif*. Ngược lại, *đoạn chương trình 2* sẽ được biên dịch.

Dạng 3:

#ifndef tên macro

Đoạn chương trình

#endif

Nếu tên macro chưa được định nghĩa thì trình biên dịch sẽ biên dịch đoạn chương trình nằm giữa #ifndef và #endif. Ngược lại, đoạn này bị bỏ qua.

Dạng 4:

#ifndef tên macro

Đoạn chương trình 1

#else

Đoạn chương trình 2

#endif

Nếu *tên macro chưa được định nghĩa* thì trình biên dịch sẽ biên dịch đoạn *chương trình 1* nằm giữa *#ifndef và #endif*. Ngược lại, *đoạn chương trình 2* được biên dịch.

Trong quá trình làm việc, thông thường các lập trình viên tự xây dựng cho mình các hàm hữu ích để phục vụ cho các công việc trong tương lai và khi cần đến có thể dùng chỉ thị ***#include*** để gắn vào chương trình chính tạo thành chương trình hoàn chỉnh. Tuy nhiên, sẽ có vấn đề nếu các tệp chương trình này được ghép nhiều lần vào một chương trình (*chương trình sẽ báo lỗi*). Để khắc phục điều này, người ta thường sử dụng các *chỉ thị biên dịch có điều kiện* nhằm tổ chức ra các tệp thư viện. Mỗi tệp thư viện như vậy cấu trúc như sau:

```
#ifndef tên macro
    #define tên macro
    /*Nội dung thực sự của tệp đặt ở đây*/
#endif
```

Theo cấu trúc này, mỗi khi tệp thư viện được ghép vào một chương trình nào đó, nó sẽ tự định nghĩa ra một *macro*. Đến lần sau, nếu trong chương trình lại tồn tại một chỉ thị ***#include*** tệp này thì khi đó trình biên dịch bỏ qua không thực hiện *ghép lần 2* tệp này vào tệp chương trình gốc nữa (*do tên macro đã được định nghĩa và đoạn lệnh nằm trong #ifndef và #endif bị bỏ qua*). Tất cả các tệp thư viện của Turbo C đều có cấu trúc theo dạng này.

PHỤ LỤC III. MỘT SỐ HÀM THÔNG DỤNG TRONG NGÔN NGỮ LẬP TRÌNH C

1. Các hàm chuyển đổi dữ liệu

1.1. Hàm chuyển đổi tổng quát sprintf và sscanf

Các hàm này được khai báo trong **stdio.h** và có dạng như sau:

int **sprintf**(char *Buf, const char* ĐiềuKhiển, danh sách đối);
int **sscanf**(const char*Buf, const char* ĐiềuKhiển, danh sách đối);

Công dụng: Các hàm làm việc giống như hàm ***printf*** và hàm ***scanf***, nhưng thay vì đưa dữ liệu *theo định dạng* ra màn hình hàm ***printf*** đưa chuỗi kết quả ra vùng nhớ ***Buf*** (*dưới dạng chuỗi kí tự*), còn hàm ***sscanf*** sẽ nhập dữ liệu từ vùng nhớ ***Buf*** (*dưới dạng chuỗi kí tự như một dòng vào*) biến đổi rồi lưu vào các biến nhớ tương ứng. Hai hàm này được sử dụng khá hữu ích để chuyển đổi các dạng dữ liệu từ chuỗi sang số và ngược lại, ghép các chuỗi với nhau...

1.2. Các hàm chuyển đổi khác

Các hàm sau đây dùng để chuyển đổi dữ liệu từ một kiểu này sang một kiểu khác rất hữu ích trong khi viết các chương trình ứng dụng.

Các hàm sau được khai báo trong ***ctype.h***

a) Hàm tolower()

Dạng hàm:

int tolower(int c);

Dùng để đổi **c** từ chữ hoa sang chữ thường

b) Hàm toupper()

Dạng hàm:

int toupper(int c);

Dùng để đổi **c** từ chữ thường sang chữ hoa

Các hàm dưới đây khai báo trong stdlib.h

2. Các hàm xử lí chuỗi kí tự

Các hàm xử lí chuỗi trong ngôn ngữ lập trình C được khai báo trong string.h và bao gồm các hàm hữu ích sau:

2.1. Hàm strchr()

Dạng hàm:

char *strchr(char *s, int kt);

Dùng để tìm lần xuất hiện đầu tiên của kí tự kt trong chuỗi s, nếu tìm thấy hàm cho địa chỉ của kí tự tìm được trong s, ngược lại hàm cho giá trị NULL.

2.2. Hàm strrchr()

Dạng hàm:

char *strrchr(char *s, int kt);

Dùng để tìm lần xuất hiện cuối cùng của kí tự kt trong chuỗi s, nếu tìm thấy hàm cho địa chỉ của kí tự tìm được trong s, ngược lại hàm cho giá trị NULL.

2.3. Hàm strcmp()

Dạng hàm:

int strcmp(char *s1, char *s2);

Dùng để so sánh hai chuỗi s1 và s2 với nhau, hàm cho:

- Giá trị âm nếu s1 nhỏ hơn s2
- Giá trị dương nếu s1 lớn hơn s2
- Bằng không nếu s1 bằng s2

2.4. Hàm strcmpi()

Dạng hàm:

int strcmpi(char *s1, char *s2);

Dùng để so sánh hai chuỗi s1 và s2 với nhau (không phân biệt chữ hoa và chữ thường), hàm cho:

- Giá trị âm nếu s1 nhỏ hơn s2
- Giá trị dương nếu s1 lớn hơn s2
- Bằng không nếu s1 bằng s2

2.5. Hàm strcspn()

Dạng hàm:

int strcspn(char *s, char * s_con);

Hàm cho độ dài đoạn đầu (*lớn nhất có thể*) của chuỗi s, mà mọi kí tự của đoạn đó không có mặt trong chuỗi s_con.

2.6. Hàm strspn()

Dạng hàm:

int strspn(char *s, char * s_con);

Hàm cho độ dài đoạn đầu (*lớn nhất có thể*) của chuỗi s, mà mọi kí tự của đoạn đó đều có mặt trong chuỗi s_con.

2.7. Hàm strlen()

Dạng hàm:

int strlen(char *s);

Hàm cho độ dài của chuỗi s.

2.8. Hàm strlwr()

Dạng hàm:

char * strlwr(char *s);

Mọi chữ hoa trong s được chuyển thành chữ thường.

2.9. Hàm strncat()

Dạng hàm:

char *strncat(char *s_nhan, char *s, int n);

Dùng để bổ sung n kí tự đầu tiên của chuỗi s vào sau chuỗi s_nhan.

2.10. Hàm strncpy()

Dạng hàm:

char* strncpy(char *s_nhan, char * s_gui, int n);

Dùng để sao n kí tự đầu tiên của chuỗi s_gui vào vung nhớ do s_nhan quản lí.

2.11. Hàm strnset()

Dạng hàm:

char* strnset(char *s, int c, int n);

Kí tự c được gán cho n kí tự đầu tiên của chuỗi s .

2.12. Hàm strrev()

Dạng hàm:

char *strrev(char *s);

Dùng để đảo ngược các kí tự trong chuỗi s, hàm cho địa chỉ của chuỗi đã đảo.

2.13. Hàm strstr()

Dạng hàm:

```
char *strstr(char *s, char *s_con);
```

Dùng để tìm lần xuất hiện đầu tiên của s_con trong chuỗi s, nếu tìm thấy hàm cho địa chỉ của chuỗi con tìm được trong s, ngược lại hàm cho giá trị NULL.

2.14. Hàm strupr()

Dạng hàm:

```
char * strupr(char *s);
```

Mọi chữ thường trong s được chuyển thành chữ hoa.

3. Các hàm toán học

Các hàm sau được khai báo trong stdlib.h

3.1. Hàm abs()

Dạng hàm:

```
int abs(int x);
```

Hàm trả về trị tuyệt đối của một số nguyên x.

3.2. Hàm labs()

Dạng hàm:

```
long int  labs(long int x);
```

Hàm trả về trị tuyệt đối của một số nguyên dài x.

3.3. Hàm rand()

Dạng hàm:

```
int rand(void);
```

Hàm trả về một giá trị ngẫu nhiên từ 0 đến 32767.

3.4. Hàm random()

Dạng hàm:

```
int random(int n);
```

Hàm trả về một giá trị ngẫu nhiên từ 0 đến n-1.

3.5. Hàm srand()

Dạng hàm:

```
void srand(unsigned seed);
```

Khởi đầu bộ ngẫu nhiên bằng giá trị seed.

3.6. Hàm randomize()

Dạng hàm:

```
void rand(void);
```

Khởi đầu bộ số ngẫu nhiên bằng một giá trị ngẫu nhiên.

Các hàm sau đây khai báo trong math.h

3.7. Hàm acos()

Dạng hàm:

```
double  acos(double x);
```

Tính arc cosine của x

3.8. Hàm asin()

Dạng hàm:

```
double  asin(double x);
```

Tính arc sine của x

3.9. Hàm atan()

Dạng hàm:

```
double  atan(double x);
```

Tính arc tangent của x

3.10. Hàm cos()

Dạng hàm:

```
double  cos(double x);
```

Tính cosine của x

3.11. Hàm cosh()

Dạng hàm:

```
double  cosh(double x);
```

Tính cosine hyperbolic của x

3.12. Hàm exp()

Dạng hàm:

```
double  exp(double x);
```

Tính e mũ x

3.13. Hàm fabs()

Dạng hàm:

```
double  fabs(double x);
```

Tính giá trị tuyệt đối của một số thực x

3.14. Hàm fmode()

Dạng hàm:

```
double  fmode(double y, double x);
```

Tính phần dư dấu phẩy động của phép chia y/x

3.15. Hàm log()

Dạng hàm:

```
double  log(double x);
```

Tính logarit tự nhiên của x.

3.16. Hàm log10()

Dạng hàm:

```
double  log10(double x);
```

Tính logarit cơ số 10 của x.

3.17. Hàm pow()

Dạng hàm:

```
double  pow(double y, double x);
```

Tính y mũ x.

3.18. Hàm sin()

Dạng hàm:

```
double  sin(double x);
```

Tính sine của x.

3.19. Hàm sqrt()

Dạng hàm:

```
double  sqrt(double x);
```

Tính căn bậc hai của x.

3.20. Hàm tan()

Dạng hàm:

```
double  tan(double x);
```

Tính tangent của x.

PHỤ LỤC IV. BẢNG MÃ ASCII TIÊU CHUẨN

Hexa-decimal	0	1	2	3	4	5	6	7
0	<NULL> 0	<DLE> 16	<SP> 32	0 48	@ 64	P 80	` 96	p 112
1	<SOH> 1	<DC1> 17	! 33	1 49	A 65	Q 81	a 97	q 113
2	<STX> 2	<DC2> 18	" 34	2 50	B 66	R 82	b 98	r 114
3	<ETX> 3	<DC3> 19	# 35	3 51	C 67	S 83	c 99	s 115
4	<EOT> 4	<DC4> 20	$ 36	4 52	D 68	T 84	d 100	t 116
5	<ENQ> 5	<NAK> 21	% 37	5 53	E 69	U 85	e 101	u 117
6	<ACK> 6	<SYN> 22	& 38	6 54	F 70	V 86	f 102	v 118
7	<BELL> 7	<ETB> 23	‘ 39	7 55	G 71	W 87	g 103	w 119
8	<BS> 8	<CAN> 24	(40	8 56	H 72	X 88	h 104	x 120
9	<HT> 9	<EM> 25	) 41	9 57	I 73	Y 89	i 105	y 121
A	<LF> 10	<SUB> 26	* 42	: 58	J 74	Z 90	j 106	z 122
B	<VT> 11	<ESC> 27	+ 43	; 59	K 75	[91	k 107	{ 123
C	<FF> 12	<FS> 28	, 44	< 60	L 76	\ 92	l 108	| 124
D	<CR> 13	<GS> 29	- 45	= 61	M 77	] 93	m 109	} 125
E	<SO> 14	<RS> 30	. 46	> 62	N 78	^ 94	n 110	~ 126
F	<SI> 15	<US> 31	/ 47	? 63	O 79	_ 95	o 111	<DEL> 127

Chú ý:

Ta có thể đọc bảng mã theo cách như sau:

- Chữ số nhỏ ở góc phải dưới thể hiện mã của kí tự trong hệ cơ số 10 (*hệ thập phân*).

- Muốn biết mã của kí tự trong hệ Hexa ta ghép thứ tự cột và thứ tự hàng của chính kí tự đó.

Ví dụ : Với kí tự ‘A’ sẽ có mã trong hệ thập phân là *65* và mã tương ứng trong hệ Hexa là *0x41*.

TÀI LIỆU THAM KHẢO

1. Phạm Văn Ất: Kĩ thuật lập trình C, Cơ sở và nâng cao. NXB KH&KT. 1996.
2. Nguyễn Văn Ba: Phân tích và thiết kế hệ thống thông tin. Các phương pháp có cấu trúc. Nhà xuất bản Đại học Quốc gia Hà Nội, 2003.
3. Lê Văn Doanh, Trần Khắc Tuấn: 101 thuật toán và chương trình. Nhà xuất bản khoa học và kĩ thuật, 1995.
4. Huỳnh Tấn Dũng, Hoàng Đức Hải: Bài tập ngôn ngữ C (từ A đến Z). NXB Lao động – Xã hội, 2003.
5. Nguyễn Xuân My, Bùi Thế Tâm: Bài tập lập trình Pascal. Nhà xuất bản thống kê,1997.
6. Scott Robert Ladd: C++ kĩ thuật và ứng dụng. Công ty cổ phần tư vấn và dịch vụ KHKT - SCITEC, 1992.
7. Đỗ Xuân Lôi: Cấu trúc dữ liệu và giải thuật. NXB KH&KT, 1995.
8. Peter Norton: Advanced C Programming. Brady Publishing, 1992.
9. Đỗ Phúc và những người khác (biên dịch): Kĩ thuật lập trình Turbo C. Công ty máy tính Việt Nam, trung tâm đào tạo tin học CMT, 1992.
10. Tống Đình Quỳ: Ngôn ngữ lập trình C^{++} dành cho sinh viên (2 tập). NXB Thống Kê, 2000.
11. Nguyễn Thanh Thuỷ và những người khác: Ngôn ngữ lập trình C thật là đơn giản, NXB Giáo dục, 1996.
12. Nguyễn Tô Thành, Dương Viết Thắng, Nguyễn Thanh Tùng: Bài tập Pascal. Trường đại học Bách khoa Hà Nội. Tài liệu lưu hành nội bộ, 1995.

MỤC LỤC

Trang

Chương 1. TỔNG QUAN VỀ NGÔN NGỮ LẬP TRÌNH C

Chương 2. BIỂU THỨC

Chương 3. TỔ CHỨC CHƯƠNG TRÌNH VỀ MẶT DỮ LIỆU

Chương 4. HÀM VÀ TỔ CHỨC CHƯƠNG TRÌNH VỀ MẶT CẤU TRÚC

Chương 5. TỆP VÀ CÁC THAO TÁC VÀO RA

Chương 6. ĐỒ HOẠ

Chương 7. CÁC BÀI TẬP TỔNG HỢP

PHỤ LỤC

Chịu trách nhiệm xuất bản:

Chủ tịch HĐQT kiêm Tổng Giám đốc NGÔ TRẦN ÁI

Phó Tổng Giám đốc kiêm Tổng biên tập VŨ DƯƠNG THỤY

Biên tập :

DƯƠNG TRỌNG BẰNG

Trình bày bìa:

TÀO THANH HUYỀN

Sửa bản in:

DƯƠNG TRỌNG BẰNG

Chế bản:

TIÊU KIM CƯƠNG

GIÁO TRÌNH NGÔN NGỮ LẬP TRÌNH C

In 3.000 bản khổ 16cm x 24cm, tại Công ty Cổ phần in Phúc Yên
Số xuất bản 65/115 - 04 CXB
In xong và nộp lưu chiểu tháng 8 năm 2004.



NGÔI SAO BẠCH KIM
CHẤT LƯỢNG
QUỐC TẾ

TÌM ĐỌC GIÁO TRÌNH DÙNG CHO CÁC TRƯỜNG ĐÀO TẠO HỆ TRUNG HỌC CHUYÊN NGHIỆP - DẠY NGHỀ CỦA NHÀ XUẤT BẢN GIÁO DỤC

(NGÀNH ĐIỆN TỬ - TIN HỌC)

1.	**Linh kiện điện tử và ứng dụng**	*TS. Nguyễn Viết Nguyên*
2.	**Điện tử dân dụng**	*ThS. Nguyễn Thanh Trà*
3.	**Điện tử công suất**	*Trần Trọng Minh*
4.	**Mạch điện tử**	*TS. Đặng Văn Chuyết*
5.	**Kĩ thuật số**	*TS. Nguyễn Viết Nguyên*
7.	**Kĩ thuật điều khiển**	*.Vũ Quang Hồi*
8.	**Kỹ thuật xung - số**	*TS. Lương Ngọc Hải*
9.	**Điện tử công nghiệp**	*Vũ Quang Hồi*
10.	**Toán ứng dụng trong tin học**	*PGS. TS. Bùi Minh Trí*
11.	**Nhập môn tin học**	*Tô Văn Nam*
12.	**Cấu trúc máy vi tính và vi xử lý**	*Lê Hải Sâm - Phạm Thanh Liêm*
13.	**Hệ các chương trình ứng dụng (Window, Word, Excel)**	*GVC. Trần Viết Thường - Tô Văn Nam*
14.	**Cơ sở dữ liệu**	*Tô Văn Nam*
15.	**Lập trình C**	*GVC Tiêu Kim Cương*
16.	**Cấu trúc dữ liệu và giải thuật**	*PGS.TS. Đỗ Xuân Lôi*
17.	**Cài đặt và điều hành mạng**	*TS. Nguyễn Vũ Sơn*
18.	**Phân tích thiết kế hệ thống**	*GVC. Tô Văn Nam*
19.	**ACCESS và ứng dụng**	*TS. Huỳnh Quyết Thắng*
20.	**Sử dụng Corel Draw**	*Nguyễn Phú Quảng*
21.	**Bảo trì và quản lí phòng máy tính**	*Phạm Thanh Liêm*
22.	**Kinh tế và quản trị doanh nghiệp (kinh tế và TCQLSX)**	*TS. Ngô Xuân Bình - TS. Hoàng Văn Hải*

Bạn đọc có thể tìm mua tại các Công ti Sách - Thiết bị trường học ở các địa phương hoặc các Cửa hàng sách của Nhà xuất bản Giáo dục:

Tại Hà Nội : ***25 Hàn Thuyên, 81 Trần Hưng Đạo, 187 Giảng Võ, 23 Tràng Tiền.***
Tại Đà Nẵng : ***15 Nguyễn Chí Thanh.***
Tại Thành phố Hồ Chí Minh : ***104 Mai Thị Lựu, Quận 1.***





Giá: 17.500đ